AN INTEGRATED APPROACH TO KNOWLEDGE SHARING AMONG HETEROGENEOUS RULE-BASED SYSTEMS

By

JONG H. PARK

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1990

ACKNOWLEDGEMENTS

I am deeply indebted to Dr. Stanley Su for introducing me to the subject area of my doctoral research, for the long and invaluable hours devoted to guidance and discussion, for serving as advisor, and for chairing this supervisory committee.

I am indebted to Dr. Sham Navathe and Dr. Herman Lam for serving on this committee and for their encouragement and guidance both professional and personal. I am grateful to Dr. Jose Principe and Dr. Paul Fishwick, for serving on this committee, and for encouragement and technical contributions to this research.

I owe a debt of gratitude to Sharon Grant for her tireless efforts to provide a well-administered research environment and for her much-valued friendship and support. I thank all my colleagues for their support and friendship, especially Rahim Yaseen for many fruitful discussions on diverse subjects.

I express my gratitude to my son, Eugene, and my wife, Soon, for their sacrifice and encouragement. They had to go to bed the countless nights waiting for me. I thank my parents, the eternal beacon of my life.

This research is supported by the National Science Foundation grant #DMC-8814989, and the Navy Manufacturing Technology Program through the National Institute of Standards and Technology

(formerly National Bureau of Standards) grant #60NANB4D0017. The implementation is supported by the Florida High Technology and Industrial Council grant #UPN-88092237.

TABLE OF CONTENTS

Abstract of Dissertation Presented to the Graduate School of the University of Florida in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

AN INTEGRATED APPROACH TO KNOWLEDGE SHARING AMONG HETEROGENEOUS RULE-BASED SYSTEMS

By

JONG H. PARK

August 1990

Chairman: Stanley Y.W. Su
Major Department: Electrical Engineering

To specify knowledge in the forms of rules and constraints has been widely accepted as a powerful methodology to extend the traditional databases. Rules and constraints can preexist, just as data can exist prior to the introduction of any integrated system over them. An approach to the integration of heterogeneous rule-based systems has been developed and the results are presented in this dissertation.

Our approach achieves the integration of heterogeneous knowledge derivation systems not at the system level, but at the rule level. That is, this approach allows the best of relevant rules and constraints to be selected in a global perspective and be executed regardless of which component systems these selected rules and constraints belong to.

This integration approach consists of three parts: a global knowledge representation scheme, a global knowledge manipulation language, and a global knowledge processing mechanism. In the global knowledge representation, the dynamic aspects of knowledge such as derivational relationships and restrictive dependencies among data items are graphically modeled by a Functional Graph to uniformly represent the capabilities (or knowledge) of the rule-based systems, while the usual static aspects such as data items and their structural interrelationships are modeled by an objected-oriented data model. For knowledge manipulation, high-level operations different from the usual data retrieval and storage operations are developed. For deriving the best global answers for high-level queries, the global knowledge processing mechanism allows the rules and constraints in different component systems to be indiscriminately exploited despite the incompatibilities in their inferencing mechanisms and interpretation schemes. One advantage of this approach is that rules and constraints can in effect be shared among heterogeneous rule-based systems. As a result, this integration approach allows the independent component systems to exchange their knowledge as well as data as if they were parts of a single system. The integration approach has been verified by an implementation.

CHAPTER I

INTRODUCTION

1.1 Motivations for the Integration of Heterogeneous Rule-Based Systems

There are a variety of applications [MCD82, BUC85, FRE85, SCH85, FIS86 and SRIN87] in which it is inefficient or even impossible to store information in the form of explicit data. As a solution, capturing knowledge in the form of rules (and constraints as special kinds of rules) has drawn considerable attention [SHO76, WON84, BRO85, SU85 and STO87]. Advanced information systems that use rules to specify knowledge include expert systems (ESs) [DEN86 and JAC86], deductive databases (DBs) [KEL82, JAR84, MIS84, BOR84 and TSU88] and database management systems (DBMSs) with constraint management facilities [KERS84, STO86, RAS87 and PIR89]. Expert systems have gained wide acceptance as sophisticated decision support systems based on artificial intelligence (AI). Deductive DBMSs have been developed to benefit from a synergic coupling of two originally-independent areas, i.e., expert system and database management system, while database management systems with constraint management capabilities are emerging as enhanced

forms of conventional database management systems.

Rule-based systems often reveal reasoning capabilities comparable to those of human experts in some aspect of a complex domain, e.g., an aircraft manufacturing company [FRE85]. However, the expertise of such a system tends to be limited to a narrow aspect. Thus, a complex environment would use a variety of expert systems working on diversified aspects, e.g., a structural analysis ES (SE), a material ES (ME), and a cost analysis ES (CE) for the design and manufacturing of a product. These rule-based systems can preexist just as independent DBMSs can exist before a distributed DBMS is introduced to integrate those preexisting DBMSs. However, they are often developed independently of each other probably at different times according to ever-changing needs and ever-evolving technologies. Consequently, they are not designed to interact with each other even though such interactions are essential for obtaining the best solution in a global perspective. Still, their expertises are interrelated with respect to their common problem domain [TRO85, TAI86, TANG86, BON87, SRIN87, QUA87 and BAN87]. Such component systems together would form a heterogeneous community of knowledge derivation systems. In such a heterogeneous environment, each component rule-based system has its own inferencing mechanism to apply to, and interpretation of, its rules, and also its own (implicit or explicit) underlying database (DB) so that it functions as an independent problem

solver. It has full knowledge to solve problems within its own aspect, but it constitutes partial knowledge with respect to the entire problem domain. To obtain a complete solution in this environment, partial solutions by its component rule-based systems must be "patched up" in an ad hoc manner if an integrated system is not available. Such a manual compilation of partial results is inevitably an error-prone process. A more serious drawback is that the complete solution from such a process is not guaranteed to be the best solution obtainable from the knowledge available in the environment. In this context, we need an integrated system for deriving the best solutions to global problems from a heterogeneous community of rule-based systems. Specifically, our integrated system should allow (1) the knowledge of independent rule-based systems to contribute to a global solution in an interactive manner, and (2) the partial solutions by autonomous rule-based systems to be assembled in an optimal way into a complete solution.

To elaborate these two points, the expertises of the member rule-based systems of a community tend to be interrelated with each other along the phases and facets of the problem domain. To make an optimal use of knowledge in such a community, the expertises of member rule-based systems must be combined in an interactive way on demand from a global perspective. For example, the SE, ME and CE in the example domain should be activated at right times along mechanical

design processes to undertake the structural analysis, the material analysis and the cost analysis of the design tasks, respectively. This cooperation mode is similar to the concurrent and interactive way in which a group of human experts usually work together, i.e., consulting each other, if necessary, for only the required portions of expertises.

There must be a systematic way of scheduling the independent rule-based systems involved in solving problems and of evaluating partial results for the resolution of conflicts and the consideration of complementary results. The systematic way should be handled automatically wherever possible. If inevitable, the users should be provided with well-defined opportunities for their decisions on very narrow parts of problem domain (e.g., on the value of one data item) so that they need not assume the responsibility of considering wide scopes of their problem domain.

1.2 Problem Statement

With the above objective in mind, we develop an approach to the integration of knowledge in heterogeneous rule-based systems, each of which is specialized only in one of many aspects of a complex global domain. By integration of knowledge, we mean that this approach allows knowledge in the form of rules and constraints to be shared among heterogeneous

component systems (CSs) despite the incompatibilities in their inferencing mechanisms and interpretation schemes.

Conventional information systems are relatively simple so that it would be rather straightforward to identify their capabilities (at a component system level) and combine their results into a global answer [HEI85 and JAC86]. However, they do not lend themselves to a close integration because the knowledge in these systems are indistinguishably mixed with their control information. On the contrary, the capabilities of rule-based systems are often sophisticated, but their underlying inferencing mechanism for deriving knowledge is by design separated from their knowledge per se captured in rules and constraints. Consequently, it is feasible to pursue a close integration of heterogeneous rule-based systems at the rule level instead of at the component system level. That is to say, at each moment of a global query processing, the best of many available rules from many different component expert systems can be chosen regardless of which component expert systems they belong to. It is a contrast to the loosely-coupled approaches [LAN77, SMI81, LAR85, CHAL86 and NAV89] where each preexisting component system is treated as a "black-box" with only its input and output "exposed" for data (or knowledge) exchange. The resulting integrated system will provide the user with a unified, global view over heterogeneous deductive DBMSs and ESs as if these systems were not independent systems but were a single system [CER85].

Thus, the users are allowed to issue queries against this integrated system as against a heterogeneous database. In this work, however, we introduce a high-level query language which provides the users with more exploratory types of queries than the usual database queries. A "super" expert system can be built on top of this system to add more functionaly such as providing global explanations.

To substantiate this integration approach, we need to know what knowledge each component knowledge derivation system has, and how to activate, and communicate with these preexisting systems to derive the best overall answer to a global query. Specifically, we develop four system components: 1) a global knowledge representation scheme to model the knowledge of preexisting rule-based systems, 2) a set of high-level operations for global knowledge manipulation to express global queries, 3) the associated global knowledge processing mechanism, which identifies, based on the global knowledge representation, the best rules with respect to a global query, activates the component systems associated with these rules in a proper order, compiles and optimizes their local results, and assembles them into a global answer, and 4) a mapping scheme and an interfacing mechanism for exchanging knowledge with preexisting component systems with arbitrary user interfaces.

The representation of knowledge, unlike data, requires that the heterogeneous system be capable of capturing the

"dynamic" relationships among data items [THO86] in addition to the data items and their structural interrelationships captured by conventional heterogeneous DBs [LAN77, ROT80, FER83 and CER855]. Specifically, the dynamic relationships specify what data items are required to derive a data item, and what data items would be affected by a change on some data item. The global knowledge, which is equivalent in effect to the union of knowledge of all component rule-based systems, is represented in our system by a functional graph. For manipulating the global knowledge, high-level queries other than the usual retrieval and storage operations [CER85 and DAT86] (e.g., gauging global impact due to local update) are introduced. In query processing, the global knowledge derivation is performed by an orderly execution of sequences of rules and constraints, which are not explicitly specified in a query, but identified through a search of the graph. As a result, unlike conventional query processing, search of rules and constraints is an indispensable global query processing technique for our approach to identify required rules and constraints along with involved entities as well as proper component systems to derive the needed data. The search also allows the best of relevant rules (in general, the best sequence of rules) or their best results to be selected at every stage of query processing and the proper component systems for executing the rules to be determined. Two special query optimization techniques are employed for such

selections.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows. Chapter Two provides a brief survey of the distributed paradigms relevant to our approach. In the light of the shortcomings of the existing paradigms, Chapter Three presents the conceptual basis of our integration approach to heterogeneous rule-based systems. We discuss how knowledge from arbitrarily related rule-based systems that form a heterogeneous community can be abstracted into a uniform global framework regardless of their incompatible inferencing mechanisms. Chapter Four outlines the overall architecture of the integrated system to be designed on the basis of our approach as described in Chapter Three, and introduces its major system components. Chapter Five through Chapter Seven develop the major systems components of our integrated approach, as outlined in Chapter Four.

Chapter Five describes the knowledge representation scheme. We show how dynamic aspects as well as static aspects of knowledge can be captured in a global perspective. We present two global knowledge (and data) representation models, which are used to uniformly capture knowledge as well as data in a heterogeneous community of rule-based systems.

Chapter Six presents the global knowledge manipulation language (GKML.) We show how complex queries can be expressed in the GKML. The high-level semantics of the GKML are illustrated with typical example queries. We define four types of new operations to satisfy the needs in complex problem domains.

In Chapter Seven, a global query processing mechanism is developed with respect to those high-level operations. To answer each type of operation, we develop algorithms for making use of the knowledge captured by the knowledge representation scheme.

In Chapter Eight, an implementation effort for validating our approach is described. We describe how the knowledge representation scheme and the global query processing mechanism as designed in Chapter Five and Chapter Seven, have been verified through the implementation.

Chapter Nine summarizes the research presented in this dissertation and suggests areas for future research. The research described here develops schemes and mechanisms for integrating heterogeneous rule-based systems so that preexisting knowledge derivation systems can function as a collaborative group in solving complex global queries.

CHAPTER II

A SURVEY OF EXISTING PARADIGMS RELEVANT TO OUR APPROACH

In this Chapter, we first discuss what characteristics our approach should have, and then we conduct a comparative survey of some existing distributed paradigms with respect to the general characteristics.

Our approach should provide a distributed problem solving paradigm for the integration of heterogeneous rule-based systems. First, our integrated system should handle distributed knowledge, at least logically and possibly geographically. By distributed knowledge, we mean that the problem solving capability of a community belongs to many independent component systems in the community. Second, this system should not be a mere data management system, but a knowledge derivation system in the sense that its component systems derive knowledge using intensional knowledge components such as rules. For this purpose, our approach has to take into account the dynamic aspects of knowledge as well as static aspects [SU85]. Third, our system is designed to take the primary responsibility of solving problems, i.e., it should be able to identify, and order the potential systems

for solving given problems. This system is a contrast to systems which only assist the user in deciding which component systems to invoke to solve complex problems. Fourth, it is to be a problem solving system rather than a data processing system in that its primary objective is to distribute a complex problem among the component systems according to their capabilities, rather than to balance workloads among them [FRE85]. Fifth, our approach pursues an integration of preexisting systems, that is, a tight coupling of existing information systems. It can be contrasted with loosely-coupled approaches [GER82, LAR85, TANG86 and KAE86] in which information systems are allowed to interact with each other only through their input and output parameters. Clearly, such an approach does not allow close cooperations among its component systems. Instead, our approach should allow the interactions among component systems to take place ubiquitously over any portion of their knowledge. That is, the knowledge units such as rules and constraints of each independent systems should be in effect exchangeable, as needed, with other systems. As a result, their integration is accomplished in the rule level rather than in the system level.

There are numerous factors that characterize a paradigm for distributed systems. In the light of our goal stated above, we consider only three of those factors such as the

component systems, control mechanisms, and integration levels. For the component systems, their properties determine the characteristics of a paradigm. Also, whether or not they are homogeneous is an important factor. Second, the control mechanism includes the strategies for describing who is responsible for the coordination among component systems, and what procedures to follow to achieve such a cooperation. Third, the integration level indicates the degree of coupling among component systems. Now, we evaluate several existing paradigms in terms of the above factors in order to solve our integration problem.

Several research efforts on heterogeneous Distributed Database Management Systems (DDBMSs) address many technical issues required to handle the heterogeneity of component systems [LAN77, ROT80, FER83, HEI85 and CZE87]. However, they deal with data only, not knowledge (i.e., extensional data as well as intensional data specified by rules and constraints). There has been a growing interest in incorporating rules with data under the name of Expert Database System (EDBS) [KERS84, BOR84, MIS84 and TSU88]. This type of systems shares some interest with our approach to the extent that it attempts to extend a data management system into a knowledge derivation system to handle intensional data as well. However, main efforts in EDBS are still grappling with issues on how to graft a stand-alone database system with a rule processing system. A few exceptions are found in works by Wilson et al.

[WIL84] and Raschid [RAS87].

Two models have been introduced in Artificial Intelligence (AI) to solve complex problems relying on multiple autonomous problem solvers. Similar to our target environment, none of their component systems has sufficient knowledge to solve the entire problem. One model is called Distributed AI (DAI) [SRID87]. There are variations within DAI in terms of cooperation methods, degree of autonomy of component systems, or granularity of knowledge [SMI81, GER82 and LES83]. However, their basic control structures are decentralized and knowledge sources (equivalent to component systems) are loosely-coupled. The other model has been developed as an extension of the conventional expert system model [TOM86]. The problem solving strategy of that model is to select the most appropriate inferencing mechanism among several mechanisms according to the current situations in the working memory. Its architecture is based on a centralized control to invoke a proper inferencing mechanism and the associate rule set. However, these two models are useful only to develop new systems, but not to integrate preexisting systems.

The Federated Information system approaches [HEI82, LYN84, HEI85, CZE87 and NAV89] are emerging with the same objective as we are aimed at, that is, the integration of preexisting heterogeneous information systems. However, it can not achieve a tight integration that our approach aims at, in

that it considers the inputs and outputs of component systems as the only two contact points among them. It regards whole component systems rather than individual rules and constraints as the atomic units of cooperation. Additional approaches with similar objectives are found in [MAR84, LAR85 and KAE86]. However, they are designed mainly to just assist the user to identify the capabilities of existing information systems, rather than to take a primary responsibility in answering user queries.

CHAPTER III

BASIS FOR THE INTEGRATION OF HETEROGENEOUS RULE-BASED SYSTEMS

In this chapter, we discuss the basis of our integrated system for preexisting rule-based systems. To realize our proposed approach, we first find some commonality among component systems as a premise for their integration. Then, we determine the unit of knowledge exchange to define the level of integration.

The existence of a certain degree of commonality is a precondition for pieces of information to be eligible for any meaningful integration among them. Information systems in general have numerous aspects which we may consider as a basis for their integration. The references to the same entity, or the same input parameters, for example, implies some possibility for integration. The schema integration problem has drawn a considerable attention as the demands are growing for independent databases to be able to be accessed as one unified database. The efforts on that problem, however, have been directed only to the factual components of knowledge [ELM79 and BAT86]. In distributed DBMSs, (extensional) data items requested in global queries can be directly fetched from

local DBs. Unlike conventional database systems, however, knowledge base systems usually deal with intensional data such as rules and constraints as well as extensional (or stored) data. In databases, the objects of integration are obvious enough to be identifiable with minor efforts.

In contrast, the integration of rules and constraints might appear to be meaningless or even impossible at a first glance. Rules, however, must go through knowledge derivation processes to generate desired data from the stored data. Individual rules of an expert system, for example, usually are not "universal" knowledge units in the sense that they cannot produce useful knowledge independently of their underlying expert system. Also, one derivation process may trigger other processes successively. As a result, the objects against which queries are issued would no longer be confined to the data items specified in the queries.

Hence, it would be impossible for an expert system E_A to import rules from another system E_B and execute them in E_A. This limitation stems from the fact that each knowledge systems has its own inferencing mechanisms for, and interpretations of, knowledge components. To integrate general expert systems that contain both kinds of data, we need to redefine the atomic unit of work into which a query is to be partitioned (it is individual data items in distributed database management systems.) To exchange knowledge captured in rules, we need to take into account the knowledge

representation schemes and control mechanisms of the information systems as well as involved data items.

With the background described above, we present the basis of our approach to the integration of knowledge specified in rules and constraints. Let's take an analogy from electric measurement, in which an engineer infers a pattern, or regresses a formula of the electric signal from intermittent sampling of its data using probes. In such a measurement, the conclusion is based only on the initial and some intermediate values of the signal, but is not concerned with the actual progress between each pair of samples of the signal. This methodology gives a clue to identify some commonality among heterogeneous knowledge derivation systems.

Application of this methodology to our approach is based on two observations.

The first observation is that a knowledge derivation process in an information system can be represented by an articulated path. A path consists of many segments. A segment denotes the subpath between two adjacent articulations. An articulation of a path refers to the junction point between its two neighboring segments. An articulation corresponds to a sampling point in the measurement example above. Specifically, an articulation of a knowledge derivation path refers to a point, at which external data may be injected to the path, or intermediate results from the path may be probed.

The second observation is that the ultimate goal of using

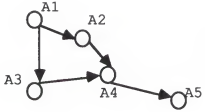
an information system is to attain desired (factual) data (and maybe associated actions.) This implies that the actual processes to derive them are not of the users' concern. These two observations can be combined into one that a knowledge derivation process can be represented by an articulated path, whose segments are delimited by their source and target data sets. A source data set of a segment denotes the parameter set that a preceding segment provides to a trailing one, while a target parameter set represents the data to be produced by the trailing segment based on the source data set.

However, different knowledge representations exhibit different frequencies of articulation. A system based on a knowledge representation with frequent articulations is referred to as a high-granularity system. The granularity of knowledge denotes the minimum size of knowledge of an information system that can be modularized without modifications on its system structure. In a rule-based system, each rule forms a segment. Specifically, the entities in its premise and consequence parts form the two articulations by which a segment is delimited.

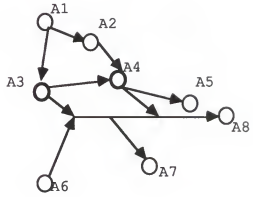
These observations lead us to naturally determine on the commonality among heterogeneous rule-based systems, and the unit of their integration. Data is apparently the only globally meaningful knowledge component across heterogeneous component rule-based systems, which they can meaningfully share with each other. Thus, data is naturally chosen as the

commonality for their integration. Also, each segment of knowledge derivation path is chosen as the atomic unit of integration.

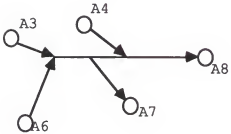
In fact, the integration can happen among chains of rules, each of which starts at their common entity (i.e., an articulation) and ends at another common entity. Each chain may traverse its own intermediate path to derive data for the same entity. Still, those chains could be regarded to be identical in terms of functionality. From this viewpoint, each expert system can be regarded as a black-box with "taps" through which data may be exchanged, as illustrated in Fig. 3-1 (a) and (b) for two component systems, CS1 and CS2, respectively. Then, they can be integrated through data items, A_3 and A_4, as shown in (c). Notice this approach is contrast to that in "loosely-coupled" systems such as Federated Information Systems, which do not consider the intermediate junction points.



(a) Knowledge Derivation Paths in CS2



(c) Integrated Knowledge of CS1 and CS2



(b) Knowledge Derivation Paths in CS1

Ai: articulations CSi: Component Systems

Fig. 3-1 Basis of Integration of Heterogeneous Expert Systems

CHAPTER IV

DESIGN OF THE INTEGRATED SYSTEM

In this chapter, we introduce the architecture, and outline the major components, of our integrated heterogeneous rule-based system. We first describe the overall architecture of the integrated system. Then, we list the features of its major components. The design of the system is governed by following two principles: 1) to achieve the closest integration possible among preexisting rule-based systems, and 2) to keep (existing) systems as intact as possible in order to maintain their autonomy.

Our integrated system needs three system components: a global knowledge representation scheme, a global knowledge manipulation language and a distributed knowledge processing mechanism. The knowledge representation is concerned with the global knowledge representation scheme by which the information required for our integration, e.g., the knowledge (or capabilities) of heterogeneous component systems, can be represented. The global knowledge manipulation language part is required as the global carrier of user requests for knowledge processing. The knowledge processing part refers to

the distributed knowledge processing mechanism by which global queries can be answered by exercising proper component systems according to their capabilities.

4.1 Overall Architecture

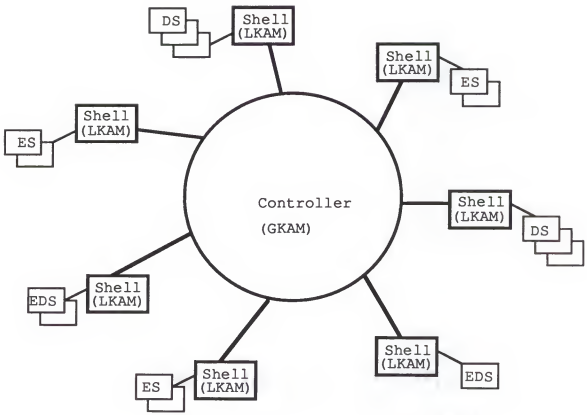
It is expected that various kinds of meta-knowledge [CER85] are needed to deal with autonomous information systems as the component systems in our integrated system. Such meta-knowledge does not belong to any of its component systems, so that we need a global controller to supervise the independently-working component systems. Still, each component system should be able to maintain its autonomy in processing tasks assigned to it. That is, the component systems should be able to preserve their own system structures in terms of knowledge representation scheme and inferencing mechanism [SRID87]. Thereby, existing systems would not have to compromise their original (maybe optimal) system structures to become component systems of our integrated system. To realize this control structure, our global control scheme has to be highly immune to the idiosyncracies of the internal control mechanisms of component systems. This control scheme allows the user to think of our integrated system as a meta rule-based system with all the knowledge from its component systems.

As shown in Fig. 4-1, our integrated system is organized in two levels of controllers: a Global Knowledge Administration Module (GKAM) and multiple Local Knowledge Administration Modules (LKAMs). This topology is similar to those of heterogeneous DDBMSs [LAN77, FER83, CER85, KRI87 and SU88] and the Federated Information System models [NAV89], but is a contrast to those of DAI models [SRID87]. Note that there are no direct interactions among LKAMs so that remote requests must be forwarded to the GKAM. As the global manager, the GKAM coordinates the activities of LKAMs in order to process user queries.

Each LKAM functions in effect as a foreman on behalf of the GKAM to process the subqueries assigned. Specifically, the functions of LKAMs are responsible for accepting and interpreting user queries, dispatching queries, which are judged beyond the local capability, for global processing, and collecting and returning to the GKAM the answers of those queries. For those functions, they perform various mapping required for exchanging information with the GKAM. An LKAM can supervise any number of component systems running on the same operating system.

4.2 Global Knowledge Representation Schemes

To handle heterogeneous component systems, the knowledge



DS: Database System
EDS: Expert Database System
ES: Expert System

Fig. 4-1 Architecture of Our Integration Model

representation scheme should be able to uniformly represent their knowledge regardless of the incompatibilities in their inferencing mechanisms and interpretation mechanisms. Yet, their knowledge derivation paths should be able to be precisely captured to achieve a close integration to meet the two principles mentioned above.

The global knowledge representation scheme for our integrated system, unlike for the conventional distributed database systems, needs to represent knowledge as well as data. Thus, we need a global knowledge representation model in addition to the conventional data representation model. Knowledge expressed in rules is not stored explicitly so it must be derived through knowledge derivation processes before it becomes usable by users. To represent knowledge expressed in rules, we need to model knowledge derivation paths according to the basis described in Chapter Three. Thus, our knowledge representation model should be able to model in a uniform manner the relevant data items and their interrelationships as specified in the rules from different component rule-based systems. Still, the knowledge representation model should be precise enough to depict the knowledge derivation paths to accomplish a close integration. Also, we need some information on how to invoke preexisting rule-based systems, which usually are not equipped with formal query languages.

Our approach is aimed to allow the query issuer not to

be concerned with the complicated knowledge derivation processes. Thus, we need a global data representation model that provides a global view of only the data items from the component rule-based systems. The relevant data items should be organized according to their structural interrelationships to provide a natural context over the entire problem domain. Thus, our global data representation model should be able to capture a variety of semantic properties among entities [SMI77 and HUL87].

4.3 Global Knowledge Manipulation Language

A Global Knowledge Manipulation Language (GKML) is a global carrier for local queries [LAN77, DWY83 and FER83]. A relative advantage of using a global KML is that mappings and translations need not to be performed between each pair of local data (knowledge) manipulation languages, otherwise the number of translators would grow with the square of the number of component systems [CER85]. Instead, mappings and translations need to be done only between each local data manipulation language and the global knowledge manipulation language. This allows the number of translators to grow only linearly with the number of component systems.

To accommodate a variety of high-level semantics that may occur in complex problem domains we are aimed at, our GKML

should be equipped with more powerful operations than the usual retrieval operations. Such non-conventional semantics include hypothetic situations and non-deterministic conditions, and may require constructs that allows for iterative processing to satisfy a given target condition. Also, the global GKML should allow the user to formulate his requests in a declarative way, regardless of the kind of data, extensional or intensional, as pointed out in the principles for the design of our system.

4.4 Distributed Knowledge Processing Mechanism

To deal with preexisting rule-based systems, our knowledge processing mechanism consists of the global processing part and local processing part. The global processing part is concerned with which component systems are to be invoked in what order to obtain the best answers to global queries. Specifically, its function includes the coordination of the exchange of data among component systems, the evaluation of the partial answers produced by those systems, and the assembly of them into the global answer. The local processing part handles mainly how tasks assigned to each component system is to be transformed into proper local queries.

A query in our declarative GKML would specify only its

involved parameters even though its underlying dependencies among those parameters might be quite complex. To fill the gaps between a tersely formulated query and its full semantics, the processing mechanism of our system should be much more involved compared to the processing of the conventional data manipulation operations. Also, our global processing mechanism should be designed to be neutral to the internal inferencing mechanisms of component rule-based systems in order to accommodate heterogeneous rule-based systems.

CHAPTER V

REPRESENTATION OF KNOWLEDGE

In this chapter, we describe the knowledge representation scheme which is concerned with representing information required for the integration of preexisting rule-based systems. To use such autonomous knowledge derivation systems as the component systems of our integrated system, we need to know what knowledge each of the component systems has and how they are interrelated in terms of their knowledge. To develop a knowledge representation scheme for the integration of such heterogeneous rule-based systems, many unique problems as well as the general issues in conventional distributed data processing systems [LAN77, ROT80, HAS82, HEI85, CER85, BAT86, DAT86 and DEM89] are to be considered.

In Section 5.1, we discuss what information is required for our approach. In Section 5.2, we present a general scheme on how the identified information can be represented. Section 5.3 and Section 5.4 describe in detail the embodiment of the general knowledge representation scheme in terms of the static and dynamic aspects of knowledge, respectively. In Section 5.5, we briefly discuss the relationship between the two

aspects of knowledge in the knowledge representation scheme.

5.1 What Information is Required for our Approach

We categorize the required information into the information on individual component systems and meta-information needed for their integration.

5.1.1 Knowledge of Individual Component Systems

Knowledge in rule-based systems are derived through inferencing using rules and applying constraints. By modeling their inferencing processes, the knowledge derivation paths in rule-based systems can be represented. To model inferencing processes in preexisting component systems, we need to consider some control information in addition to their expert knowledge [THO86]. Expert knowledge refers to the actual problem solving knowledge as opposed to the control information needed simply for implementing such knowledge. The control information of a component system relevant to our approach includes the information on how to query the component system and how to exchange data in the course of inferencing.

As a basis of our research, we investigated the rule structures of a variety of representative rule-based systems

such as MYCIN systems [SHO76 and BUC85], OPS5 systems [MCD82 and BRO85] and Prolog systems [STE86 and TSU88]. We first discuss what aspects of expert knowledge of individual component systems are required to be captured. Then, the requirements on the control information are discussed.

1) Expert Knowledge

According to the basis of integration described in Chapter Three, the expert knowledge of rule-based systems are to be represented in terms of articulated paths. To represent articulated paths, we need to identify the entities (or data items) which form the articulations in the inferencing chains, and their relationship, across a heterogeneous community of rule-based systems.

The relevant entities constitute variables contained in rules and constraints. Besides an attribute of an object class [BUC85 and BRO85], an operation associated with an object class, e.g., DELETE, and an object class including one to represent a relationship among object classes, e.g., MARRIAGE between MALE and FEMALE, also can be an entity in our approach. Besides the relevant entities, we need to capture the structural relationships among the involved entities, which provide the users with a global semantic view over the entire problem domain [SU90]. The structural interrelationships are often abstracted into aggregation,

generalization, interaction, and so on [HAM81 and HUL87]. We call relevant entities and their structural interrelationship the static aspects of knowledge.

Another required aspect of expert knowledge is the dynamic relationships among entities. This aspect of expert knowledge represents the derivational relationship or the restrictive dependency among entities. For a deductive rule, it specifies what entities are to be instantiated to derive a value on some entity. Another aspect is the control flow information in general, the triggering condition in particular for rule-based systems. This aspect specifies under what condition a rule (or constraint) is to be activated to derive new data or to enforce the intended dependency among involved entities. These three are the rudimentary aspects of knowledge (called the dynamic aspects of knowledge) to be required for our approach to describe inferencing paths in component rule-based systems.

Besides the above two basic aspects of knowledge, we consider several specific aspects of expert knowledge with respect to their relevancy to our approach. We restrict the aspects of knowledge required for our approach to those globally meaningful and having direct effects on sharing knowledge with other systems. In some constraint management systems, timing options may be attached to constraints to tailor their triggering mechanism. In RAS87, for example, the PRE, PARA and POST options determine the triggering time of

constraints. Those options also are irrelevant to our approach in that their effects will be confined within their associated constraints (and their associated component systems.) As other constructs of knowledge, Fuzzy Set Concepts or Confidence Factors are not considered since their effects are limited within the system using them.

Whether the condition clauses of a rule are connected conjunctively or disjunctively is an important part of expertise. This distinction, however, is not relevant to our approach because all the clauses need to be checked in either case, so both types of connections are uniformly treated in our knowledge representation.

2) Control Information

A rationale behind our approach is to make our integrated system independent of the internal control structures of particular systems. To that end, our approach is designed in such a way to minimize the required control information in each component system. Thus, we capture only information in component systems that has direct impacts in interacting with other component systems (or the external world.)

First, we need to distinguish whether or not the value of an entity in a component system is to be input from or output with the external world. The value of an entity may be explicitly stored in an underlying database, or derived as an

intermediate result in an inferencing process depending on the user interface of a system. This distinction on each entity is required to determine how to exchange data with related component systems with respect to the entity.

Another essential control information is the I/O protocols of each component systems, which correspond to query languages in DBMSs. Unlike in DBMSs, the input and output in rule-based systems are performed through arbitrary user interfaces instead of using formal query languages. Thus, we need a new query translation scheme and data exchange method for issuing queries to such systems. The mode of chaining [RIC83] is also added to our essential control information, considering that it affects the I/O procedure in terms of the query translation in each component system.

On the contrary, the classification of rules, for example, is irrelevant to our approach in that it is a trick merely for a better internal efficiency [RAS87]. For the same reason, knowledge specified by meta-rules [BUC85, NII86 and JAC86] also are considered as irrelevant control knowledge.

5.1.2 Information for Integration

Meta-information in our approach refers to the information required to integrate the knowledge of independently-running rule-based systems into a cooperative group of rule-based systems. That is, given a problem beyond

the capabilities of any one component system, some information would be needed to decide on which component systems and how to cooperate to solve it.

In our approach, two expertises (parts of expertises, precisely) may be interrelated in one of three patterns: precedence, parallelism and no relation. For the precedence pattern, the precedence information has to be specified among interrelated expertises. Such information would provide the order in which they are to be invoked to collectively produce a global answer. Two expertises in parallel relation may be alternative or complementary in terms of their functionality. For an alternative type, some information is required to select the best among parallel expertises. Specifically, some measures are required for quantifying their relative performances in terms of, e.g., accuracy and efficiency. Also, a certain global criterion is needed to resolve conflicts that may occur among values derived by multiple expertises about the same problems. For the first two patterns of relationships, it also needs to be indicated that there exists some relationship between them. Even in case no relationships exist among expertises, that "null" relation as such can be a useful information.

For all the information mentioned above, however, our integrated system is not expected to automatically handle every situation. As a consequence, various deadlocks will occur at run-time calling for human interventions, so that

some facility for user intervention is an integral provision in our integrated system. We shall see that one of our query types allows the user to interactively steer the query processing to his specific needs.

5.2 How to Represent Knowledge

The various kinds of information described so far, are all organized in two frameworks of knowledge representation. The first framework is the Global Database Schema (GDS) which provides an integrated (data) view over all the Local Database Schemas (LDSs) of the component systems [CER85]. Besides the GDS, our integrated system, unlike the other distributed DBMSs, needs a global knowledge schema for capturing the dynamic aspects of knowledge.

Conceptually, the two aspects of global knowledge are organized into two tiers of schemas over the actual knowledge sources, namely, the component rule-based systems, as illustrated in Fig. 5-1. The Global Database Schema in the upper tier provides a unifying global external (or query issuers') view over the entire problem domain as used in conventional heterogeneous DBMSs [LAN77, FER83 and CER85], while the lower tier of global knowledge representation is required for internal processing of knowledge to answer global queries [CER85]. The second schema

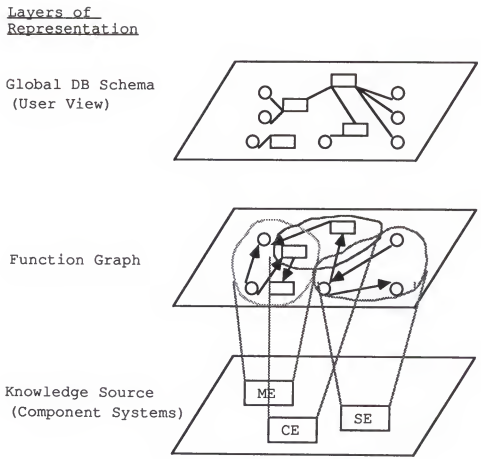


Fig. 5-1 Architecture of Knowledge Representation

is invisible to the user.

5.2.1 Required Characteristics of Knowledge Representation Models

The knowledge captured in the GDS is basically the static aspects of expert knowledge such as relevant entities, their structural interrelationships, and their association with component systems. To capture such static aspects of knowledge as discussed previously, a high-level data representation model is desirable. That is, the model should be equipped with rich semantic constructs enough to be able to capture various kinds of entities and their structural interrelationships. In this light, we use the Object-Oriented Semantic Association Model (OSAM*) as our data representation model. In collaboration with its object-oriented concepts, its various association types offers an abundant menu of constructs for capturing a wide range of semantic properties [SU90]. Thus, its inherent semantics could provide the user with a natural view over the entire problem domain, which would not be possible with a conventional data model [TSI76, STO76 and DAT86].

The behaviors (or dynamics) of information systems have been attempted to be modeled in such ways as transaction modelling and behavior modeling [BRO81 and SAK83]. But, such efforts have been directed to the development of new systems.

In contrast, our model should be able to represent the behaviors of existing systems with reasonable efforts. For this purpose, we need a knowledge representation model to model the dynamic aspects, which are not captured by the GDS but is integral for our integration. Some of such aspects belong to expert knowledge, and the others to control information.

5.3 Representation of Static Aspects of Knowledge

The GDS provides query issuers with a uniform global view over a heterogeneous community of rule-based systems. The entities in the GDS are captured along with their semantic interrelationships. Thus, users can issue proper queries since they command a global semantic view over the entire domain of discourse. We first introduce our underlying global data representation model by which we can represent the static aspects of knowledge comprising relevant entities and their structural interrelationships. Then, we touch on the integration of LDSs into a GDS focusing on the unique issues on our integration procedures.

5.3.1 Object-Oriented Semantic Association Model (OSAM*)

The Object-Oriented Semantic Association Model (OSAM*)

provides a conceptual basis for uniformly capturing the semantics and interrelationships among objects in a complex application domain [SU85, SU86, SU88 and SU90]. These objects can be physical objects, abstract concepts or events. An object is represented in an OSAM* database as an object identifier (OID) and is described by a set of descriptive data. Objects are grouped together into classes based on some common semantic properties they share.

In OSAM*, there are two types of object classes: entity object class (or E-class) and domain object class (or D-class.) Homogeneous sets of objects of interest in a problem domain are modeled by E-class. A D-class, on the other hand, models the structure and behavior of objects that primarily serve as descriptive data of other E-class or D-class objects. The fundamental difference between an E-class and a D-class is that an E-class contains a set of instances, which are explicitly created by the user of the database. Each instance has a corresponding object identifier (OID) assigned to it by the DBMS. Whereas, a D-class is a domain of possible values which are used to describe objects modeled by some other object classes. The values in a D-class identify the objects and thus no OID's are assigned.

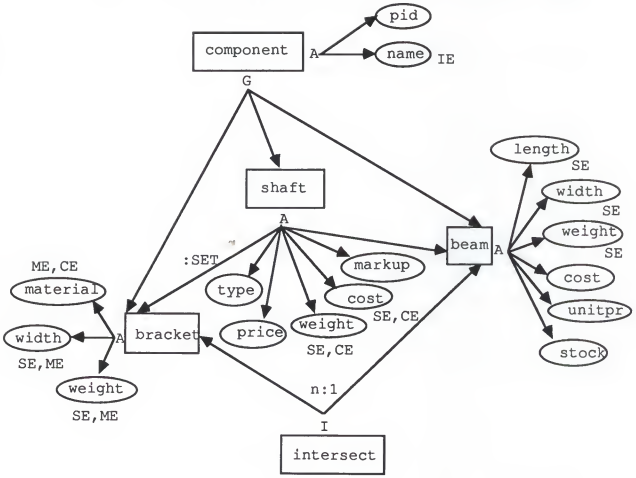
An association type in OSAM* defines a semantic relationship between or among a set of object classes. There are two categories of association types in OSAM*: semantic association types and data constructors. A semantic

association type connects an object class called defined class to a number of object classes called constituent classes. There are five semantic association types defined in the OSAM* data model, namely, Aggregation (A), Generalization (G), Interaction (I), Composition (C), and Cross-Product (X). A data constructor serves to define a complex domain, which is a structure of some other defined domains. Examples of data constructors are Set, Vector, Matrix, Time-Series, Ordered-Set, Bag, etc. The domains they form are sets of integers, vectors of reals, bags of items, etc.

5.3.2 An Example OSAM* Database Schema

The query examples in the remainder of this paper are based on an example problem domain. The example domain is concerned with designing and manufacturing mechanical shafts. A shaft is composed of a set of brackets and a beam, which may intersect with each other. The semantic diagram (S-diagram) in Fig. 5-2 shows an OSAM* database schema corresponding to the example problem domain.

We describe the problem domain in terms of the characteristics of its component systems. The example problem domain consists of three rule-based systems, i.e., Structural Analysis Expert System (SE), Material Engineering Expert System (ME) and Cost Analysis Expert System (CE). Their knowledge is represented by the rule bases are shown in Fig.



* ME,SE,CE,IE: Association with CSs

Fig. 5-2 A Global Database Schema for Example Domain

5-3. The three expert systems deal with different aspects of their common problem domain, i.e., mechanical design. However, each system has its own rule format, inferencing mechanism and interpretation scheme. SE is a forward-chaining system on structural analysis; ME is a forward-chaining system on material engineering; CE is a backward-chaining system on cost analysis. The three component systems cover various categories of rule-based systems. Specifically, ME could be viewed as a constraint management system to the extent that all the entities handled in the system are explicit entities. CE provides a framework for integrating deductive database management systems in that its queries are similar to ordinary retrieval operations. SE functions as a conventional expert system [MCD82].

An S-diagram is an OSAM* representation which provides a graphical representation of classes and their associations by means of a network of labeled vertices and edges. Some commonly recognized constraints associated with the classes are also specified in the S-diagram.

In the S-diagram, vertices stand for classes. A rectangular vertex represents an entity class (E-class), and a circular vertex represents a domain class (D-class). A vertex that represents a defined class is connected to its constituent classes via labeled edges. Edges are grouped together in the S-diagram if they have the same types of association with the defined class and are labeled by the

Rule Set for Structural Analysis ES

```
SR1: If BEAM.LENGTH=x and BEAM.WIDTH=y
     then BEAM.WEIGHT=x*y.
SR2: If BEAM.WEIGHT=x and BRACKET.WEIGHT=y
     then SHAFT.WEIGHT=func1(x,y).
SR3: If BRACKET.WIDTH=x and BRACKET.MATERIAL=y
     then BRACKET.WEIGHT=10*x*y.
SR4: If SHAFT.WEIGHT=x then BEAM.PRICE=100*x.
SR5: If BEAM.PRICE=x then SHAFT.PRICE=x+100.
SR6: If BEAM.WEIGHT>100 then BEAM.COST=100.
```

Rule Set for Material ES

```
MT1: If COUNT(BRACKET)>10 then EXECUTE MR1.
MR1: If SHAFT.WEIGHT>100
     then BRACKET.MATERIAL="IRON"
     else BRACKET.MATERIAL="COPPER".
MR2: If SHAFT.WEIGHT>100 then INSERT(BRACKET).
```

Rule Set for Cost Analysis ES

```
CR1: If SHAFT.VOL>10 then SHAFT.COST+SHAFT.MARKUP=100.
CR2: If SHAFT.COST=x and SHAFT.MARKUP=y
     then BEAM.PRICE=10*x*y.
```

Fig. 5-3 Rule Sets of Component Systems (CSs)

semantic association type (A, G, I, C, or X). Edges are labeled by A, I, and X represent the attributes of a defined class. The constituent classes to which these edges connect are the domains from which the attributes draw their values. The generalization association is similar to the superclass-subclass concept of object-oriented systems.

To explain the example S-diagram briefly, the SHAFT is described by several attributes such as two E-classes BEAM and BRACKETs and five D-classes, TYPE, PRICE, etc. The object classes, BRACKET, SHAFT and BEAM are organized into a generalization hierarchy (denoted by a G) with the super-class, COMPONENT. The class, INTERSECT, is used to represent an interaction that can exist between two object classes, BRACKET and BEAM. In the figure, such an interrelationship is represented by an interaction association (denoted by an I.) The associated CSs are indicated beside each data item, e.g., ME and CE at BRACKET.MATERIAL, in the figure, meaning that those CSs use the data item.

5.3.3 Construction of Global Schema

A global database schema is constructed through a schema integration process [BAT86]. A schema integration refers to a process through which local database schemas are merged into a global database schema. It encompasses the addition of new local database schemas to an existing global database schema.

The integration of conventional database schemas has naturally focused on data, so it has been considered in terms of only object class, relationship, and structural constraint. In our approach, however, a resulting global database schema would include entities involved in rules and constraints as well as stored data. For the control information, the global database schema keeps track of which component system each entity belongs to.

As the first step in the schema integration, the entities from all the component systems have to secure their global identities along with their associations with object classes. For that purpose, each component system has to prepare an OSAM* schema of its own underlying database. In reality, however, many rule-based systems such as expert systems do not have their (explicit) underlying databases. For these systems, the integrated system would have to create new schemata for their "implicit" databases, while for those with underlying databases, their existing database schemata have only to be transformed into OSAM* versions of schemata. After that, each local schema goes through a general integration process to become part of the global database schema. We assume that the global database schema can be formed by the Global Knowledge Administrator. This schema creation and integration is just an ordinary database modeling problem [HEL88], which is beyond the scope of our research. The schema integration is an extensive research topic on its own [ELM79, BAT86 and CZE87].

5.4 Representation of Dynamic Aspects of Knowledge

We model the knowledge in component rule-based systems in terms of knowledge derivation paths and some relevant control information. As the framework for this modeling, we need a knowledge representation model that allows knowledge derivation paths to be abstracted into their relevant entities and the dynamic interrelationships among those entities. As a result, the knowledge derivation paths available across a heterogeneous community of rule-based systems should be able to be uniformly captured. The relevant entities are usually explicitly specified in rules in the form of the arguments of the functors in Prolog [STE86] or the parameters in MYCIN [SHO76], for example.

We first give a formal definition of our knowledge representation model. It is developed in a stepwise fashion. Then, a brief description on its graphic notations and terminology are given.

5.4.1 The Knowledge Representation Model

In this subsection, we develop the Function Graph (FG) as our knowledge representation model. We define the elements of the Function Graph and discuss how the required aspects of

knowledge can be captured in the Function Graph. As a similar graph, the Constraint Graph is used to graphically represent constarints. However, the Function Graph is different from the Constraint Graph [LEL88] in several aspects in that the Function Graph: 1) deals with the representation of the knowledge from existing rule-based systems, 2) captures the knowledge derivation paths rather than the operator and operands relationships, 3) contains various meta-information for the integration of heterogeneous rule-based systems, and 4) deals with general rule-based systems such as deductive database and rule-based expert system as well.

The global knowledge in a network of rule-based systems is the union of the knowledge of these component systems. The knowledge of each rule-based system is specified in the form of rules (deductive rules or constraints.) A rule is the smallest unit of behavior in a rule-based system [KERS84, SU85, TRO85, THO86 and BON87]. Consequently, a rule becomes the atomic unit of our integration scheme and the basic unit of our knowledge representation. We first describe how the knowledge specified by each individual rule can be represented in our approach. Based on the basic representation scheme, we show how the knowledge of each component system, and then the "global" knowledge of the entire heterogeneous community can be represented.

The information required for our integration consists basically of two parts [THO86, NII86 and JAC86]: the expert

knowledge, i.e., the problem solving knowledge each component system possesses, and the control information, i.e., information on how to use each component system. The control information is concerned mainly with interfacing the heterogeneous rule-based systems. We shall describe it in the context of query processing mechanism. In this section, we describe the representation of the expert knowledge.

1) Representation of the Knowledge Specified in a Rule

The objective of our integration is to allow rules from heterogeneous systems to be used as if they were all based on a single inferencing mechanism. Accordingly, we capture only the information that is globally meaningful independent of the underlying inferencing mechanisms. To capture the knowledge specified in a rule, we consider both the static and the dynamic aspects of the rule. The static aspect refers basically to the involved entities in a rule, while the dynamic aspect consists of the declarative part such as derivational or restrictive dependencies among these entities, and the procedural part such as the underlying inferencing mechanism including triggering mechanism, conflict resolution strategy and interpretation scheme. Extracting only globally meaningful information, a general rule can be abstracted into an antecedent-consequent relationship among entities (or data items) and optionally an explicit triggering condition. As we

shall show, these three items are sufficient information for our integration. That is, this abstraction allows rules from different component systems to be uniformly represented to form "global" inferencing paths (corresponding to the global knowledge from these systems.)

In general, a rule consists of three parts: antecedent part (IF part), consequent part (THEN and/or ELSE part) and optional triggering condition part. Each part in turn consists of a number of clauses, which are connected with each other either disjunctively or conjunctively. As described earlier, however, we do not distinguish how the clauses are interconnected. To represent a rule, we first examine its individual clauses.

Many (simple) facts can be expressed by a triplet, (OBJECT ATTRIBUTE VALUE), which means that ATTRIBUTE of OBJECT is equal (=) to VALUE. This form is the canonical data structure used in conventional rule-based systems in AI field [BUC85 and JAC86].

To accommodate a wide range of semantics in general rule-based systems that our approach deals with, however, we consider a generalized form of clause which allows a relational connective between an attribute of an object and its value. Besides the mere equality, such relational connectives include >= (equal to or greater than), =< (equal to or less than), != (not equal), etc., or a general relationship such as a functor in terms of Prolog. Another

generalization is to allow objects to be the arguments of a clause. Most existing expert systems do not directly support relationships among objects [SHO76, MCD82, BRO85 and BUC85], so that the rigid triplet data structure above has so far sufficed. To support the relationships among objects as well, we consider objects as possible arguments of clauses.

In our knowledge representation scheme, a clause is defined by a triplet, (E, θ, υ) where entity set E = {e| e = o.a, attribute a of object o}; connective θ = < | >= | =, etc. | predicate name (or functor name [STE86]); υ = optional constant value. As mentioned above, our approach is concerned only with E among the three elements for representing the global inferencing paths containing the rule. (Notice that υ is a special value of an element of E, or its effect like that of θ, would be confined within the clause.) As a result, a rule is abstracted into a triplet, R = (A, C, Tr), where entity set in antecedent part, $A = \cup_i E_i$, $1 \leq i \leq$ # of clauses in IF part; entity set in consequent part, $C = \cup_j E_j$, $1 \leq j \leq$ # of clauses in THEN part; entity set in triggering condition part, $Tr = \cup_k E_k$, $1 \leq k \leq$ # of clauses in triggering conditions. For a production rule in particular, Tr = A or C, depending on its associated chaining mode (forward or backward.) That is, the triggering condition for such a rule is by nature predefined in production systems as matching of its IF or THEN part without an explicit condition, thus the corresponding entity set A or C is the same as the entity set

in triggering condition Tr.

A clause (E, θ, υ) can be graphically represented by a set of vertices as shown in Fig. 5-4 (a). E_1 represents a clause consisting of e_{11} enclosed by a shaded line, and E_2 represents a clause of e_{21}, e_{22} and e_{23}. An entity can be one of three types, i.e., object class, attribute of object, or function associated with object class such as DELETE or NUMBER_OF_OBJECTS. Thus, a vertex is defined by: 1) a vertex id, 2) the vertex type, and 3) the data exchange mode:

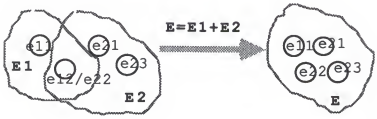
v = (id, type, mode),

where id = entity name, type = Object ¦ Attribute ¦ Function, and mode = EXplicit ¦ IMplicit. An "explicit" entity (in EXplicit mode) refers to a data item whose values are to be input from or output to the external world (depending on the user interface of the underlying system.) An entity that is not an explicit entity is called an implicit entity. Data exchange on an implicit entity requires a special interfacing scheme as will be discussed later.

The dynamic aspect of knowledge, i.e., derivation relationship or restriction dependency, between a pair of entity sets, i.e., A and C, and the associated triggering condition on an entity set, i.e., Tr, can be represented by a two-tail edge as shown in Fig. 5-4 (b) & (c). In (c), a rule (with Tr = A or C) is represented by only two sets of entities, while a constraint (with an explicit triggering condition) is by three separate sets of entities. As shown in

E1 = {e} E2 = {e1, e2, e3}

a) Clause: (E,Δ,v)



b) IF, THEN part or triggering condition: E=E1+E2+...

Fig. 5-4 Stepwise Knowledge Representations (Continued)

(d), a constraint c_i has two entities in its antecedent and consequent parts, respectively. The shaded edge pointing to the c_i represents its triggering condition. Thus, the abstracted rule R = (A, C, Tr) can in general be graphically represented by an edge with three vertex sets:

$$e = (W_a, W_c, W_t)$$

where head W_a = vertex set corresponding to A, tail W_c = vertex set corresponding to C, and tail W_t = vertex set corresponding to Tr.

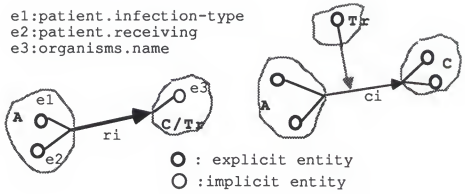
An example of a simplified version of rule from MYCIN (based on backward-chaining) might look like:

IF The type of infection is bacterial, AND
The patient is receiving colistin,
THEN There is evidence that the organisms involved are E.coli

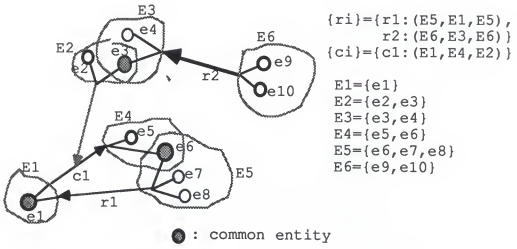
This rule consists of three clauses, (PATIENT, INFECTION-TYPE, =, bacterial), (PATIENT, RECEIVING, =, COLISTIN) and (ORGANISMS, ID, =, E.COLI), and would consequently be represented by a triplet R of ({PATIENT.INFECTION-TYPE, PATIENT.RECEIVING}, {ORGANISMS.NAME}, {ORGANISMS.NAME}), which is graphically shown in Fig. 5-4 (c).

2) Representation of the Knowledge of a Component Rule-Based System

Using the two constructs, e and v, the knowledge base of a component system i based on inferencing mechanism I_i can be



c) rule(ri) and constraint(ci): (A,C,Tr)



d) knowledge of a rule-based system: {ri} + {ci}

Fig. 5-4 Stepwise Knowledge Representations

represented by a graph (a Dependency Graph) as shown in Fig. 5-4 (d). The edges corresponding to the rules of a component system are meshed into such a graph via their common vertices, e.g., e_1, e_3 and e_6, which are indicated by dark circles in Fig. 5-4 (d). Specifically, two edges e^i_m and e^i_n are connected if their common vertex set, $V_c = (W^m_a \cup W^m_c \cup W^m_t) \cap (W^n_a \cup W^n_c \cup W^n_t) \neq \emptyset$. In general, the Dependency Graph is a directed multi-vertex graph $DG^i = (W^i, E^i)$ where $W^i = \{W^i_1, W^i_2, \ldots\}$ is a non-empty set of vertex sets, and $E^i = \{e^i_1, e^i_2, \ldots\}$ is a set of edges.

A dependency graph DG^i captures the dependencies among rules and constraints in a component rule-based system i, and consequently all possible knowledge derivation paths in the system. Specifically, a path from v_1 to v_2 means that v_2 can be derived if the sequence of rules corresponding to the path are invoked starting at v_1 according to the order specified by the path. Note, however, that it does not exactly represent the actual triggering orders in inferencing chains because the internal triggering orders within a component system is non-deterministic in the sense that they vary depending on the values in its working memory. Rather, DG^i contains all possible knowledge derivation paths in a system i. To be precise, an actual inferencing chain consists of a sequence of subgraphs, $DG^i_j \subseteq DG^i$, within each of which the order of edges correctly reflects the triggering order in the corresponding segment of the inferencing chain. The exact triggering orders among rules

in inferencing chains will be considered in the query processing part.

3) Global Representation of Knowledge of a Community of Rule-Based Systems

To represent the global knowledge of an entire community, the multi-vertex, directed graphs DG_i for all its component systems are merged via their common vertices into a multi-vertex, directed multi-graph (called Function Graph.) It is a multi-graph because, if $I_i(e_1) \neq I_i(e_2)$ where $I_i(e)$ denotes the underlying inferencing mechanism of edge e, two edges e_1 and e_2 are distinguished in a Function Graph even if $W_a(e_1) = W_a(e_2)$, $W_c(e_1) = W_c(e_2)$ and $W_i(e_1) = W_i(e_2)$. Semantically, such two edges constitute two different knowledge derivation paths (of length 1) deriving two alternative values for the same entities, $W_c(e_1$ or $e_2)$, from the same entities, $W_a(e_1$ or $e_2)$, based on two distinct inferencing mechanisms. Such an entity is represented as a "confluent" vertex, namely, one with in-degree > 1 [AHO74]. Accordingly, the definition of a vertex is extended to include another attribute, aggr, which denotes an aggregation function for resolving conflicts among alternative values. A global knowledge base administrator (KBA) should assign such functions in a global perspective. Consequently, v = (id, type, com, aggr), where aggr = MINimum ¦ MAXimum ¦ AVeraGe ¦ user INTervention. MIN, MAX and AVG mean

the minimum, maximum and average of the alternative values, respectively, while INT indicates an undefined aggregation function for user intervention. Also, an edge in a Function Graph is defined with additional information: 1) the associated inferencing mechanism, and 2) two metrics to quantify the processing efficiency of, and the accuracy of the knowledge specified by, the corresponding rule. As a result, an edge in Function Graph is defined by a tuple of six attributes:

$$e = (W_s, W_c, W_t, I, ef, af),$$

where I = id of inferencing mechanism (i.e., id of a component system), ef = efficiency factor, and af = accuracy factor. The two factors, ef and af, are used as measures for the cost evaluation of alternative (or parallel) paths. The accuracy factor indicates how accurate knowledge (or data) a rule can derive, while the efficiency factor indicates how efficiently a rule can be executed. These factors are assigned in a consistent manner across the community (again by a KBA.)

The efficiency of executing a rule depends on three major elements such as the efficiency of the algorithms used in the rule, the efficiency of its underlying inferencing engine and the performance of the computer system on which the associated system is running. In turn, two levels of time components make up the algorithm element. The first level is due to individual clause, and the other level accounts for functions embedded in each clause. For a multi-clause rule, the time components

for each clause are added to make its ef. The value of ef ranges from one to infinity. The precise definitions of the af and ef are given below.

An edge e corresponds to a rule r, which in general consists of clauses c_j and a clause c in turn may contain functions f_k. The ef for an edge e_i, ef_i, is computed as follows. $ef_i = e^s_i \times e^I_i \times [\ \Sigma_j(e^c_j + \Sigma_k e^f_{jk})\ ]$ where e^s_i denotes the system performance factor of the component system underlying e_i, e^I_i denotes the efficiency factor of the underlying inferencing mechanism, e^c_j denotes the efficiency factor of unit clause c_j with no regard to embedded functions (presumably $e^c_j \propto$ # of entities involved in c_j), and e^f_{jk} denotes the efficiency factor for function f_k embedded in c_j. While, af does not hinge on the system performance. Rather it depends only on the algorithms used in the rule. The af for an edge e_i, $af_i = \Pi_j(a^c_j \times \Pi_k a^f_{jk})$ where a^c_j denotes the accuracy factor of unit clause c_j with no regard to embedded functions (presumably $a^c_j \propto$ # of entities in c_j), and a^f_{jk} denotes the accuracy factor for function f_k embedded in c_j.

In practice, it might not be easy to quantify precisely the accuracy of an algorithm. Thus, a focal point in computing af's is how to coherently apply a unified scale to all the edges across different systems in a community. As a practical measure, these two factors are allowed to be specified on the system level in order to alleviate the efforts to compute ef's and af's. Also, we shall see that our query processing

mechanism uses another simple but reasonable scheme for estimating ef's and af's if no better information is specified.

In the Function Graph, rules (and constraints) from different rule-based systems together form knowledge derivation paths. A path of heterogeneous rules is no different in semantics from one in a Dependency Graph except that different segments of the path belong to different component systems. However, the partial inferencing chains in different systems corresponding to these segments are to be externally threaded via their common entities to properly establish a virtual global inferencing chain corresponding to the whole path.

Besides such a serial relationship among heterogeneous rules, their parallel relationship also carries significance in our integration. Parallel paths [AHO74] in a Function Graph can be viewed to have the same functionality in the sense that they can derive values for the same target parameters using values of the same source parameters. In effect, the sequences of heterogeneous rules corresponding to parallel paths are collectively exchangeable with one another despite their different intermediate entities and the incompatibilities among their individual rules' underlying inferencing mechanisms.

As a result, a Function Graph uniformly captures the

dynamic aspects, i.e., derivational and restrictive dependencies, of global knowledge, and consequently the potential paths of knowledge derivation across the heterogeneous rule-based systems. It can also be viewed as containing paths of impacts among entities. We now define the notion of Function Graph more formally below:

Function Graph, FG = (W,E,L) is a directed multi-graph consisting of a finite, non-empty set W of vertex sets, i.e., $W = \{W_i \mid W_i \subseteq V\}$ where the vertex set $V = \{v_i \mid v_i$ is an individual entity$\}$, a set of edges E and a set of links L. Let $W_i = \{w_{i1}, w_{i2}, ..\}$, then $w_{ij} \in V$ and in general $W_i \cap W_j \neq \emptyset$. $L = \{I_i, 1 \leq i \leq \#$ of component systems$\}$ represents the set of inferencing mechanisms associated with the edge set E. An edge e is an ordered tuple (W_a, W_c, W_t, I_i) where $W_c \neq \emptyset$. $E = W \times W \times W \times L = \{e \mid e = (W_a, W_c, W_t, I_i)$ where $W_a, W_c, W_t \in W$ and $I_i \in L\}$. Two labelling functions f1 and f2 are defined as f1: $(W^a \times W^c \times W^t \times L) \rightarrow EF$, and f2: $(W^a \times W^c \times L) \rightarrow AF$, where $EF = \{ef \mid ef > 0\}$ and $AF = \{af \mid af > 1\}$, and $W^{a,c,t} = \{W_{ai,ci,ti} \mid W_{ai,ci,ti} = W_{a,c,t}$ of $e_i\}$. We take $f2(W_i, W_j, I_k) = \infty$ if the relational connective θ of any clause in $W_c \neq =$.

5.4.2 Graphic Notations and Terminology

In general, a Function Graph is composed of one or more connected subgraphs (in the sense of undirected graph in

conventional term [SED83]). We describe the graphic notations for the elements of, and some important terminology for, the Function Graph.

As a graph, a Function Graph is represented by two basic elements, vertices and edges. Consider the FG shown in Fig. 5-5 corresponding to a part of knowledge specified by the rule sets of expert systems in our example domain. Vertices represent entities (or data items) of interest in the domain of discourse. The circular vertex, diamond-shaped vertex, and rectangular vertex,

represents the three types of vertices, i.e., attribute of an object class, operation of an object class and object class, respectively. A vertex name is attached to each vertex as its id. A bold vertex indicates an explicit entity, for example, mode(BEAM.WIDTH) = EX, whereas an ordinary vertex indicates an implicit entity, for example, mode(BEAM.WEIGHT) = IM. That is, BEAM.WEIGHT is not explicitly stored in a component system so its value is derived by a rule or through a chain of rules. Whether an explicit entity is an input or output parameter can be decided by whether an explicit entity is on the tail of an edge or on the head.

A solid-line edge (or bold-line edge) represents a derivational relationship (or restrictive dependency.) The vertices on the tail of an ordinary solid-line edge corresponds to an antecedent vertex set, W_a, while the vertices on its head correspond to the associated consequent

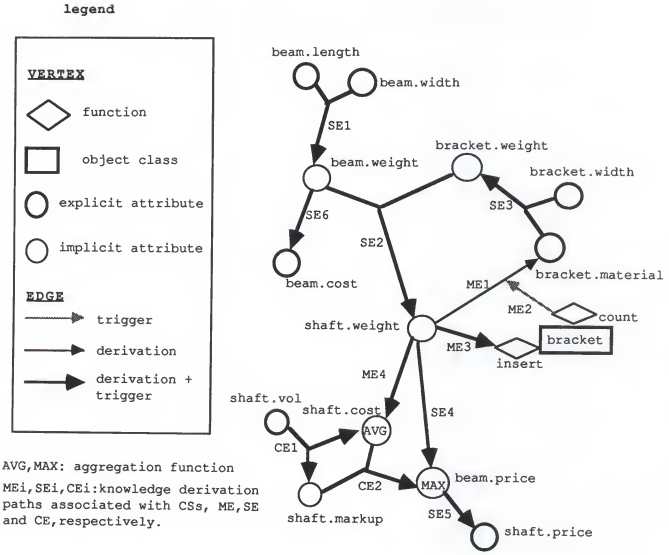


Fig.5-5 A Function Graph Corresponding to the Rule Sets Given In Fig.5-3

vertex set, W_c. The vertices on the tail of a shaded-line edge represents a triggering vertex set, W_t. A bold line indicates that a triggering vertex set coincides with either its associated antecedent vertex set or consequent vertex set, that is, a production rule. For a constraint, a shaded edge associates its triggering condition with its dependency path. For example, ME2 associates a triggering condition with the dependency path ME1. Note, however, that a triggering edge merely specifies the roles between the triggering edge and the associated constraint, but its direction does not reflect the actual order of execution between them. An edge represents the unit of knowledge derivation within individual component systems. Consequently, it is used as the unit of integration of them in our approach.

The auxiliary elements such as the associated component system, I(e), the mode of chaining, and two factors for performance evaluation, ef(e) and af(e), are indicated for each edge. Also, the conflict resolution option, aggr(v), is marked on each confluent vertex.

Specifically, the id of component system marked beside the edge specifies its associated component system. The direction of an arrowhead on the edge belly indicates the mode of chaining used in its associated component system. It is assumed to be a forward chaining unless a reverse arrowhead is explicitly put on its belly. Note, however, that the associated component system can be applied to vertices, and

the mode of chaining is meaningful only to bold edges. For aggr(e), MAX, MIN, AVG, and UND beside a confluent vertex represents maximum, minimum, average and undefined, respectively. The UND option is the default option. For ef(e) and af(e), two numbers are attached to edges, respectively.

We introduce some terminology. An edge is defined by an ordered pair of vertex sets. The edge SE1, for example, is defined with W_c = {BEAM.WEIGHT} and W_t = {BEAM.LENGTH BEAM.WIDTH}. A path in the FG is a sequence of edges in the form (W_{c1},W_{t1}), (W_{c2},W_{t2}), ..., (W_{cn},W_{tn}) such that W_{t1} intersects with W_{c2}, W_{t2} with W_{c3}, and so on. That is, a path P_i from S to T is defined as a set of edges such that P_i = $\{e_i | e_i, 1 \le i \le n$, where $W_c(e_1) \cap S \neq \emptyset$; $W_c(e_j) \cap W_a(e_{j+1}) \neq \emptyset$, $1 \le j \le n-1$; $W_c(e_n) \cap T \neq \emptyset)$. For example, there is a path from the vertex, SHAFT.VOL, to the vertex, SHAFT.PRICE, through a sequence of edges, CE2 and SE5. Two edges are connected if there is at least one path between them. A junction vertex denotes a vertex which is associated with more than one component system. For example, the vertex SHAFT.WEIGHT is a junction vertex that is associated with two member expert systems, SE and ME. In particular, a confluent vertex is a junction vertex into which multiple edges converge, e.g., BEAM.PRICE. Hence, the vertex, BRACKET.MATERIAL, for example, is not a confluent edge even though it is a junction vertex. Parallel paths denote paths whose ending vertices coincide. Specifically, two paths P_1 = $\{e_{1i} | e_{1i}, 1 \le i \le m\}$ and P_2 = $\{e_{2j} | e_{2j}, 1 \le j \le n\}$ are parallel paths if

$W = W_c(e_{1m}) \cap W_c(e_{2n}) \neq \emptyset$ for $1 \leq k \leq m$ and $1 \leq l \leq n$, and there exists at least one v such that $v \in W$ and v is on both P_1 and P_2. There may or may not exist edges such that $W_1(e_{1r}) \cap W_1(e_{2s}) \neq \emptyset$ for $1 \leq r \leq m$ and $1 \leq s \leq n$. Semantically, parallel paths represent multiple knowledge derivation paths with the same functionality in the sense that they derive values for the same entity from the same entity. This implication is an essential concept for our query processing mechanism as will be seen. The definition of a cycle in our approach also is extended according to the semantics captured in the Function Graph. A path $P = \{e_i | e_i, 1 \leq i \leq n\}$ contains a cyclic subpath if there exist edges such that, $..> e_{m-2} > e_{m-1} > e_m, .. > e_n > ..$, ($e_1 > e_2$ means e_1 precedes e_2 on a path) and $W = W_c(e_n) \cap W_1(e_{m-k}) \neq \emptyset$ where $k \geq 0$. Such a cyclic subpath P_c with respect to a path $P = \{e_i | e_i, 1 \leq i \leq n\}$ is represented by $(\{e_{m-k}, e_{m-k+1}, e_{m-k+2}, .., e_n\}, W_R)$ where reference vertex set $W_R = W$.

5.5 Relationship between Global Data Schema and Function Graph

In the preceding section, we described how the (dynamic aspects of) knowledge can be modeled. As discussed 5.2, the static aspects of knowledge, i.e., the involved entities and their structural interrelationships, are extracted to form the upper tier in the global knowledge representation. Of the entities involved in Function Graph, however, only explicit

entities (i.e., explicit objects and their explicit attributes) appear in the Global Database Schema defined by the Object-Oriented Semantic Association Model.

Global Schema, GS = (V^g, E^g) consists of a vertex set V^g and an edge set E^g. V^g = {v|mode(v)=EX) = O_{EX} ∪ A_{EX} where O_{EX} = {v| type(v) = Object and mode(v)=EX) and A_{EX} = {v| type(v) = Attribute and mode(v)=EX).

A Function Graph and the associated Global Schema are interrelated since $V^g \subseteq V$. The S-diagram in Fig. 5-2 represents a global schema that is associated with the Function Graph shown in Fig. 5-5. The two schemas capture the static and dynamic aspects of the knowledge specified by the rule sets of component systems: SE, ME and CE.

CHAPTER VI

GLOBAL KNOWLEDGE MANIPULATION LANGUAGE

6.1 General

In this chapter, we develop a global query language as the global carrier for the user requests. In DB query languages [SHI81, COD82 and ZAN83], the supported operations are specialized mainly for the management of large quantity of data. In expert systems, however, various operations required for decision support are arbitrarily embedded in their user interfaces [CHA86 and JAC86]. We attempt to enhance the query languages for data management into a (formal) query language for decision support.

In developing this enhanced query language, we focus on three features that are essential in decision making but are lacking in the existing query languages: (1) handling dynamic situations, (2) processing data items not explicitly specified in queries, and (3) repetitive processing of query [BEL82].

The primary operations in DBMSs are to retrieve or update individual data items satisfying some given conditions. In contrast, operations in expert systems are mainly to inquire

into the overall changes due to some change on a part of the (actual or virtual) underlying DB according to the interdependencies among data items. Thus, this enhanced query language should provide the user with facilities to examine the consequences of enforcing constraints and using rules. In this respect, notice the limitations imposed by the strict dichotomy between retrieval and storage operations which existing query languages deal with [SHI81, COD82 and ZAN83].

DBMSs deal only with data items explicitly specified in queries. The queries in expert systems, on the other hand, are concerned primarily with the interrelationships among an extensive range of data items. Consequently, some data items involved in a query may not be known in the querying time or too many to be specified in the query completely. Thus, the user should be allowed to issue queries with some of parameters unspecified. For example, the user may give only the "source" conditions without having to specify the "target" parameters. This property would give users a handsome freedom in issuing queries, and would enhance to a great extend the expressive power of queries.

The given target conditions in a complex query may be satisfied only after many attempts with different source conditions. For such a complex query, an unconventional facility would be required to allow the user to observe in phases the evolving results of the complex query and to accordingly adjust the source condition. Interactive session

is an integral provision for this feature [URS83].

Specifically, the enhanced query language should allow the user to:

(1) Handle dynamic situations. Operations are needed for examining the global effects due to a local cause.
(2) Operate on hypothetic conditions [GOL84]. With this feature, the user can observe the impacts due to some hypothetic condition.
(3) Operate with non-deterministic parameters. The parameters in a query may not be known in the querying time. Identifying those parameters becomes a part of query processing.
(4) Express unconventional semantics. An example of this query type would be one that allows the user to obtain a "source" condition to satisfy a given "target" condition.

In the next section, the query types supported in our language are described. In Section 6.3, the semantics of NULL value and NO value are briefly discussed. Finally, some distinctive properties of the resulting query language are described in Section 6.4.

6.2 Query Types Supported in our Global Query Language

Our query language supports four types of user queries.

Cause and effect is the general semantics across the four types of queries. In other words, the queries are aimed to gauge the changes caused by an update on some part of the underlying database.

Specifically, the following four types of queries are supported.

Type I. Get the global impact on certain data items due to some conditions.
Type II. Get the entire global impact due to some local conditions.
Type III. Get the source conditions that satisfy some target conditions.
Type IV. Get all the data items that can potentially affect certain data items, and their current values.

The query examples in the remainder of this chapter are based on the example problem domain given in Chapter Five. The queries are issued against the S-diagram in Fig. 5-2 to represent the example domain.

The processing of queries of these types in general needs to evaluate the current and the newly updated value of each entity involved in the queries. We assume in the following discussions that the user will start issuing queries with respect to some particular instance (or value) on each data item. We call those instances (or values) "current" instances

(or values.) For an engine designer, for example, the latest version of engine design should be the current instance of engine.

1) Type I

A type-I query is composed of a set of target parameters, $T = \{t_i \mid t_i$, a data item in a DB schema$\}$, and a set of source conditions, $C_s = \{c_i \mid s_i=v_i$, where s_i is a source parameter and v_i is a value to be assigned to $s_i\}$. A type-I query returns the value of each target parameter t_i (in the GET clause) resulting from applying the source conditions c_i (in the WHEN clause.) This type of query allows the user to examine whatever change that would occur on the target entities if an update is made to the source entities. The query, "What would the cost of shaft be when the length of beam is 100 feet?" can be expressed as:

```
GET shaft.cost
WHEN beam.length = 100
```

Example 1. Compute the cost of shaft when length of beam is 100 feet.

It is possible to formulate this query to return the relation of the two values of each target parameter (one value before and the other after the source conditions are applied.) The original value of target parameter is stored in a DB. In such a format, two values of a source parameter are specified

using an relrative construct, e.g., TIMES or PLUS. The query, "What would the price of shaft be in terms of ratio, if the length of beam is doubled?" would be phrased as follows:

```
GET shaft.price
WHEN beam.length = beam.length TIMES 2
```

Example 2. Compute the change in the price of shaft when the length of beam is doubled.

The second beam.length in the WHEN clause represents the original (current) value of beam.length. The answer would be expressed in terms of a ratio between the original value and the newly derived value using TIMES. The value resulting from the multiplication would become the value of the first beam.length in the condition clause. The output of this query shows whether and how much the target parameters have been affected by the given conditions. If no change is detected on a target parameter, the value of NO will be returned on the parameter.

Notice the enhanced expressive power of this type of queries. The net effect of a type-I query is approximately equivalent to those of three relational operations and some algebraic manipulations. Specifically, one retrieval, one update and one retrieval in order are to be performed with subsequent comparison operations. This nature also implies a significant advantage in developing algorithms for query processing. Namely, the existing query processing techniques

for the conventional relational operations could be exploited to process the queries in our language.

2) Type II

A type-II query is composed of a set of source conditions, $C_s = \{c_i | s_i=v_i, s_i$ is a source parameter and v_i is a value to be assigned to $s_i\}$. Unlike a type-I query, this type of query does not specify the target parameters. The target parameters are to be identified with respect to the given source parameters. This type of query returns the changed value of each of the identified target parameters affected by the conditions given in the WHEN clause. This type of query allows the user, who does not know a priori which data items to be affected, to examine all the changes caused by an update made on the source parameters. A query, "Retrieve the value of each parameter that is affected by the update that the strength of beam is increased by 10 percents." would be formulated as:

```
GET *
WHEN beam.strength = beam.strength PLUS beam.length * 0.1
```

Example 3. Collect all data items along with their values affected by the increase of the strength of beam by 10 percents.

This type of query involves non-determinism to the extent that the target parameters of query are not fixed until their

final values are derived.

3) Type III

A type-III query is composed of a set of source conditions $C_s = \{c_i | s_i = v_i$, where s_i is a source parameter and v_i is its initial value) and a set of target conditions $C_t = \{c_j | t_j \theta v_j$, where relational connective θ is =, !=, <, >, >= or =<). This type of query returns a value of each source parameter such that the source conditions can satisfy the target conditions, and the corresponding value of each target parameter. A query, "Compute the weight of bracket such that the cost of shaft does not exceed 1000 dollars.", would look like:

```
SET bracket.weight
FOR shaft.cost < 1000 WITHIN 10
```

Example 4. Set the maximum weight of bracket such that the cost of shaft is less than 1000 dollars with an error margin of 10.

If the initial value of the source parameters are not given, the current values will be used instead. The construct WITHIN specifies the error margin permissible with respect to the target condition. With error margin m, the satisfactory value v on a target parameter t_i is, $v_i - m < v < v_i + m$.

The target conditions (given in the FOR clause) usually will not be met with the initial values of the source

parameters (given in the GET clause.) Thus, this type of query may go through many "sessions", until the target conditions are met. Each session is roughly equivalent to a type-I query. In the second or later session, however, the source conditions are adjusted based on the outputs of the previous sessions. The adjustment of the values of the source parameters can be performed through either user intervention or automatic mode. With the intervention mode chosen (it is the default option), the user is offered an opportunity at the beginning of each session to adjust the values of source parameters until the target conditions are satisfied.

By specifying an increment (using a BY construct), a query of type III will be processed in an automatic mode. Then, the source conditions are to be automatically adjusted in each session by the given increment (or decrement.) In this automatic mode, the query in Example 4 will look like:

```
SET bracket.weight = 100 BY 5 %
FOR shaft.cost < 1000 WITHIN 10
```

Example 5. Set with no user intervention the weight of shaft so that the cost of shaft is less than 1000 dollars.

The semantics of this query is, "Compute the maximum weight of bracket such that the cost of shaft is less than 1000 dollars. Start the computation with 100 pounds as the initial value of the weight of bracket and adjust the source condition with an increment of 5 percents in each iteration."

An intuitive algorithm for answering a query in the automatic mode is sketched as follows. First, initial values for source parameters are fetched from the database (or given in the FOR clause). Unless the result of applying those values satisfies the target conditions, the increment specified with the BY, is added to each of the initial values prior to the next session. After such an adjustment, the next session is executed with the adjusted values as the source conditions. If the resulting values on the target parameters still are not satisfactory values, the deviation of the value of each target parameter from the value given in the target condition, is measured and compared. The results of these comparisons decide the direction, increase or decrease, of the subsequent adjustments on the values of source parameters.

Specifically, the values are to be adjusted in alternate directions at the first two sessions, but later, the directions of adjustments depend on the results in the preceding sessions. Namely, if the two latest sessions with the same increment yield values such that the deviations monotonically shrink, the subsequent adjustment should be with the same increment. However, if the deviation changes in its sign rather than in the magnitude, it implies the previous adjustment has been overdone. Then, the increment is halved for the adjustment in the subsequent session, and so on. This process repeats itself until all the target conditions are met or the number of sessions exceeds a certain limit. Note that

these tactics will work only if no divergence is involved in the results of the preceding sessions. Of course, the user reserves his option to intervene in the adjustment process at any session.

4) Type IV

A type-IV query is composed of a set of target parameters $T = \{t_i | \text{ data items in the DB schema}\}$. This type of query returns a set of source parameters along with their values, which the target parameters can be affected by or derived from. An example query "Retrieve all data items along with their values, which are related to the price of shaft." could be expressed as:

GET *
FOR shaft.price

Example 6. Find all entities and their current values to potentially affect the price of shaft.

The source parameters are not specified in type-IV queries. Rather, they are to be identified.

6.3 NULL Value and NO Value

In query types I and II, the result of a query can take two singular values such as NULL and NO. Some of the target

parameters given in the query or identified may not have been affected by the source conditions. In this case, the output value of such a target parameter is NO to mean no change has been caused by the given source conditions. On the other hand, a value of NULL means that a result does exist but happens to be a NULL value, while the interpretation of NULL value varies according to specific semantics [COD79 and ZAN83].

6.4 Properties of the Global Query Language

The resulting query language reveals the following properties:

(1) High expressive power. A query of type-I, for example, is roughly equivalent to two retrieval operations and one comparison operation.

(2) Comprehensive Semantics. The four query types cover virtually all basic query operations supported by various rule-based systems [BRO85, BUC85 and STE86].

(3) Mixture of storage and retrieval operations. A query may involve retrieval and storage operations.

(4) Repetitive operations. Some iterative query processing usually is required to satisfy the target condition specified in a type-III query.

(5) Terse syntax. The declarative expression of queries frees

the users from being concerned about the procedural (or navigational) part of queries.

(6) Easy to implement. A query can be decomposed into conventional retrieval operations (along with minor arithmetic operations.) As a consequence, most existing techniques for query processing can directly be employed to process these types of queries.

CHAPTER VII

DISTRIBUTED QUERY PROCESSING MECHANISM

In Chapter Five, we described a knowledge representation scheme by which the knowledge of a community of preexisting rule-based systems can be represented. Then, Chapter Six presented a global knowledge manipulation language in which the users can formulate their queries in a global perspective against such a heterogeneous community of knowledge derivation systems. In this chapter, we develop a distributed knowledge processing mechanism by which user queries can be answered.

The component systems of our integrated system are autonomous systems working independently of one another. Also, a typical query in our integrated system cannot be answered by any one component system, rather its processing would involve the knowledge from many component systems. The function of our query processing mechanism is to obtain the best, coherent answer for such a query from partial answers derived by its independently-working component systems. In effect, the knowledge of our integrated system is equivalent to an "organic" union of all the knowledge that its component systems possess.

To process queries using the knowledge distributed among many component systems, our query processing mechanism needs to deal with various unconventional issues with respect to identifying relevant component systems that need to be involved in answering a global query, and resolving various conflicts among their local answers. Among the various aspects of knowledge, the user in our approach sees only data items captured in the global database schema. Their interrelationships and interdependencies, which essentially constitute knowledge in our knowledge representation scheme, are invisible to the user. Also, such complex relationships are not (and need not be) specified in his/her query. Consequently, there is considerable semantic gap between the users' view (and their queries) and the actual knowledge relevant to processing those queries. As a distributed system, our integrated system would also involve many conventional issues in general distributed database system. However, we will focus on the issues unique to our approach, while we minimize the discussion on the conventional issues as possible.

We describe the query processing in two levels: global query processing and local query processing. In the global level, the query processing goes through four steps: translation of query, identifying relevant knowledge, decomposition of the query into tasks and their ordering, and execution of query. The local processing basically is

concerned with how the tasks are mapped into proper queries for their associated component systems.

This chapter is organized as follows. The overall procedure of our query processing mechanism is introduced in Section 7.1. Then, the four global query processing steps are described in detail in Sections 7.2 through 7.5. In Section 7.6, the query processing in the component systems is discussed.

7.1 Overall Procedure of Query Processing

The query processing is started by submitting a query formulated in the global knowledge manipulation language. The user issues the query in reference to the global database schema, in which only the relevant data items from the component systems have been integrated in a global conceptual model. Unless the parameters specified in a query are extensional (or stored) data items, they have to be derived by using a number of rules from, probably, many component systems. Before giving the detail of each query processing step, the four global processing steps and the local processing step are outlined as follows:

Step 1: Translating a global query. The four query types supported in our system are in essence translated into

the following formats, respectively: I. T_1 GIVEN S_1, the values of target entities $T_1 = (e_{ti})$ given values of source entities $S_1 = (s_i | e_{si}=v_i)$, II. * GIVEN S_2, the values of all entities affected by the given source condition $S_2 = (s_i | e_{si}=v_i)$, III. S_3 FOR T_3, the values of source entities $S_3 = (e_{si})$ to satisfy target condition $T_3 = (t_i | e_{si}\ \theta_i\ v_i)$ where θ_i = relational connective = | >= | etc., and IV. GET T_4, the values of source entities (unspecified in a query) for deriving T_4.

Step 2: Identifying required rules (and constraints) and their firing order across a community of heterogeneous rule-based systems for the query, i.e., identifying virtual global inferencing chains by using the Function Graph. The rules involved in a query are not specified in the query as opposed to the data items involved in the query. For this reason, relevant rules must be identified via a search of the Function Graph. In this step, the association of rules with component systems are not considered. A global query optimization may be performed based on the characteristics such as the efficiency or accuracy of rules themselves.

Step 3: Partitioning the identified rules into a sequence of independently-executable groups of rules according to their association with component systems, i.e., dividing the identified global

inferencing chains into partial inferencing chains. The identified rules will in general belong to many component systems, so they need to be partitioned into groups before each group can be executed independently of the others. The order in which the rules and consequently their associated component systems are executed is strictly determined by their interdependencies via their common entities. (In contrast, data items in DBs are mutually independent.)

Step 4: Executing subqueries corresponding to the groups of rules and combining their local answers, after a global query optimization, into a global answer [LAN77 and SU88], i.e., establishing the virtual global inferencing chains. Unlike the first optimization, this second global query optimization is performed based on the results of the execution of identified rules. That is, if more than one value on the same entity are derived by different component systems, the query processor determines which of such alternative values is the optimal one.

Step 5: Mapping of subqueries into "executable" forms in their associated component systems according to their user interfaces. Due to the nature of knowledge derivation, this mapping is different from the query translation between the global and local query languages

in conventional distributed DBs [LAN77, ROT80, FER83 and CER85]. That is, the mapping of a subquery in a rule-based system amounts to supplying the initial data [BUC85] required to start inferencing for obtaining an answer to the corresponding local query. Also, intermediate data may be requested as the inference progresses.

7.2 Translation of Query

In general, the purpose of query translation is to tokenize entities involved in the query, to decompose the query into a number of basic operations in some internal representation, and to convert to some canonical form [CER85 and DAT86]. However, the query translation in our approach deviates somewhat from ones in the conventional DBMSs. For example, the involved entities, called source and target parameters, may not be specified in the query for some query types. As a result, some of the parameters can be determined only with final results of query processing. In this light, our query translation needs to deal with some non-determinism. This deviation is due to the natures of query types supported in our approach.

We will not dwell on the detailed mechanism of query translation because it is basically the same as those in

conventional database systems [CER85, DAT86 and SU88]. Instead, a brief description of its general procedure is given to provide the input to the following query processing step. The first information to be decided in the query translation step is the type of query and its attached options, which prescribe the various strategies on subsequent query processing. Then, after a usual lexical analysis with reference to the global database schema, the source and target parameters are determined.

As described in Chapter Six, each query type has various options that direct query processing mechanism or output formats. For query type I, for example, the query output can be formatted in a relative form, i.e., expressed in terms of ratio between a newly-derived value and an existing value. Also, the processing strategies in repetitive query processing for a type-III query can be different depending on its option. We shall describe our query processing mechanism mainly for the query type I which is the basic type. For the other types of queries, only distinct issues are discussed whenever appropriate.

Basically, a query is translated into the query type, the source and target parameters, and options, i.e., (TYPE, S, T, [OPTION]), where TYPE denotes the query type, S denotes the source parameter set, T denotes the target parameter set, and OPTION denotes options attached according to the type of query. A query, "What would be the PRICE of SHAFT be if the

LENGTH of BEAM is doubled ?", for example, will be translated into a 4-tuple, (TYPE-I, (BEAM.LENGTH), (SHAFT.COST), (TIMES-2)). In the subsequent query processing steps, we will use this 4-tuple (or 3-tuple) to represent the query. In queries of types II and IV, however, the source or target parameter set is initially empty, meaning that the parameters are not determined at this step.

7.3 Identifying Required Rules and Constraints

In the query translation step, the query type (and some options), and the source and/or target parameters have been identified. With those information given as input, this step decides in a global perspective whether and how the query can be processed by the community of rule-based systems. Specifically, we identify the required rules (and constraints) in a proper firing order, and their associated entities, with respect to the two sets of parameters. Notice that the association of rules with component systems is not considered in this step of query processing. The objective of this step of query processing in effect is finding (virtual) global inferencing chains with respect to the query.

The query processing in this step is based on a search of the Function Graph. For all four types of queries,

identifying required rules, their firing order and associated entities can essentially be generalized as searching the Function Graph for paths from S to T (S in a type-III query is all the entities connected to T.) The semantics of a derivation path $p = \{e_1, e_2, \ldots, e_n\}$ from a vertex v_1 to another v_n in the FG is that the value on v_n is derivable from or is dependent on the value on v_1 in such a way that intermediate results can be obtained progressively through the sequence of the edges, $e_1, e_2, \ldots, e_n$, on the path. By progressively, we mean that the preceding edge, e.g., e_1, of each pair of adjacent edges, e.g., e_1 and e_2, is driven by the trailing edge, e.g., e_2, through their common vertices, e.g., $V = W_c(e_1) \cap W_t(e_2)$, in a pipelined fashion. That is, the rules and constraints corresponding to those edges on the paths need to be activated in that order to obtain a correct value on v_n, the target parameter, with respect to a change on v_1, the source parameters. Recall that, for constraints in general, $W_t \neq W_s$ unlike for rules. Thus, W_s and W_t need to be identified separately. In Fig. 7-1, for example, there is no direct path through ME1. In case no such a path connecting the source and target vertices for a query is found, the query processing terminates immediately with "no impact" (i.e., the NO value) as its final output.

Besides the ordinary paths described above, parallel paths and cyclic paths are two kinds of paths required to be identified for special processing in the subsequent steps.

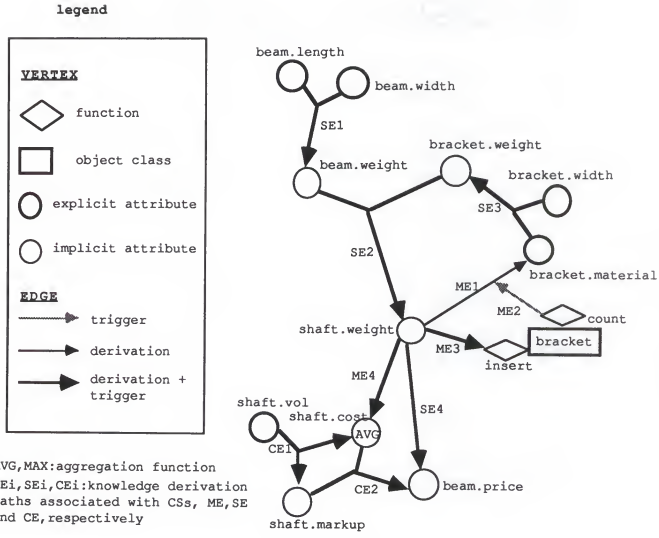


Fig. 7-1 Part of the Function Graph Given In Fig.5-5

Parallelism in Path: The existence of parallel paths implies multiple knowledge derivation paths with the same functionality in the sense that they derive values on the same entity. The two paths of ME4 and CE2, and of SE4 in Fig. 7-1, for example, constitute parallel paths from the vertex, SHAFT.WEIGHT to the vertex BEAM.PRICE. Among such parallel paths, we may select one or more paths depending on the users' requests. For such a selection, the AFs and EFs attached to edges are used as the measures for the evaluation of parallel paths. If the user specified an option in favor of efficiency, for example, in the query, the path of choice would be the most efficient path judged in terms of the EFs of the parallel paths. Paths containing a constraint are selected regardless their performance for an obvious reason due to their characteristics, i.e., specifying restrictive dependencies among entities. Note that parallel paths may not have in common their starting vertices while their ending vertices always are the same. For example, the path of SE1, SE2 and ME4, and the path of CE1, as shown in Fig. 7-1, are considered two parallel paths with respect to SHAFT.WEIGHT.

The composite EF, ef(P), of a path P is computed by accumulating the individual EFs of the component edges on the path. A composite AF, af(P), can be obtained by multiplication of all the individual AFs. Recall that $af(e) = \infty$ for an edge e corresponding to a constraint. Consequently, af(P) for a

path P containing any constraint would be computed to be ∞, so that it would unconditionally be selected in addition to the one(s) with the highest finite accuracy (i.e., containing only deductive rules.)

Cyclic Paths: The cycle presents a routine problem in many fields of computer science [AHO74 and PET85]. A cycle represents an infinite number of paths. In rule-based expert systems, it is addressed in the context of the cyclic chain of inference [BUC85]. In our approach, the cycle is expected to be a more common phenomenon in that our integrated system deals with many independently developed rule-based systems, whose rules tend to be arbitrarily intertwined in terms of their relevant entities.

In rule-based systems that are concerned with proving propositions (mostly based on backward-chained inferencing), cycles are to be detected and eliminated because they would cause only useless looping [SHO76 and BRO85]. As opposed to ones in analysis problems, however, cycles generally cannot be simply discarded since they may produce different information at different cycles as in synthesis problems [WIN84]. To accommodate general semantics, our strategy of handling the cycle is based on the trend in a series of results that a repetitive execution of a cycle would reveal.

In general, a cycle is defined to be a path of length at least one which begins and ends at the same vertex [AHO74].

In our approach, the general definition of the cycle has been extended to account for the generalized edges or multi-vertex edges used in the Function Graph. Specifically, a cyclic path in the Function Graph was defined to be a path on which at least one vertex exists in both its tail vertex set of its starting edge and the head vertex set of its ending edge. As shown in Fig. 7-1, for example, a cycle is detected along the path consisting of SHAFT.WEIGHT, ME3, BRACKET, ME2, ME1, BRACKET.MATERIAL, SE3, BRACKET.WEIGHT, SE2 and SHAFT.WEIGHT. The algorithm to deal with cyclic paths is given in the following:

Algorithm 7.3.1

1. If there is any segment $P_s = \{e_{s1}, e_{s2}, \ldots, e_{sn}\}$ of a path P, i.e., $P_s \subseteq P$, such that its starting edge's triggering tail vertex set and ending edge's head vertex set intersect, i.e., $W_c(e_{sn}) \cap W_t(e_{s1})$, put them in CYCLE_SET, i.e., CYCLE_SET $\leftarrow P_s$.

2. For each P_i in CYCLE_SET, choose as its reference vertex a vertex such that the starting and ending edges intersect, i.e., $W_R = W_t(e_{i1}) \cap W_c(e_{in})$. If more than one such vertex exist, select the first of them with respect to the path as the reference vertex.

Now we describe the general algorithm for identifying knowledge derivation paths from a source parameter set to a

target parameter set. The Function Graph is not an ordinary graph so existing graph algorithms are not directly applicable to it. To render it amenable to existing graph algorithms, a Function Graph is first converted to an equivalent ordinary graph, a normalized Function Graph or nFG. A normalized FG, nFG = (V,E^n,L), is an ordinary directed graph consisting of a set of vertices V and a set of edges E^n. $V = \cup_i W_i$, and $E^n = (\cup_i W_{ia} \times W_{ic} \times l_i$, if $W_H = W_{ia}$ or $W_{ic}) \cup (\cup_i W_H \times W_{ic} \times l_i$, if $W_H \neq W_{ia}$ or $W_{ic}) = \{e^n \mid e^n$ = a triplet (h,t,E^s) where head $h \in V$, tail $t \in V$, and the set of associated edges E^s such that $E^s \subseteq E$ and $E^s = \{e \mid t(e_n) \in W_t(e)$ and $h(e^n) \in W_c(e))\}$. That is, an edge in the nFG is defined by two vertices instead of two (or three) sets of vertices. For example, an edge $e = ((v_1, v_2), (v_3, v_4))$ in a FG would be normalized into four edges in the corresponding nFG, $e^n_1 = (v_1, v_3)$, $e^n_2 = (v_1, v_4)$, $e^n_3 = (v_2, v_3)$ and $e^n_4 = (v_2, v_4)$. The search algorithm is given below:

Algorithm 7.3.2 Search for an ordered set of paths from source parameter set S to target parameter set T

INPUT: a normalized Function Graph, nFG, and S and T, where S or T may be ø.

OUTPUT: a set of sub-graphs of edges $P = \{P_1, P_2, \ldots\}$ and their costs $c(P_i)$, where c=ef & af.

METHOD: 1) Conduct a depth-first search of nFG for all

possible paths of normalized edges e^n from S to T, 2) Calculate the composite EF and AF for each path, 3) Denormalize edges e^n into the original edge e, and 4) Eliminate duplicate paths.

```
begin

     P = ∅;
     for each u in S do
        begin
          let e^n be any edge such that h(e^n) = u;
          SEARCH(e^n) ;
        end
end

procedure SEARCH(e^n):

   begin

     i = 1;

     for each edge f^n such that t(f)=h(e^n) do
        begin

          P_i = P_i ∪ f^n;
          if h(f)∈T then
             begin

             /* getting composite ef & af for a path of
          many edges */
               ef(P_i)=Σ_i ef(e_i); af(P_i)=Π_i af(e_i);

             /* getting the best path in terms of ef &
          af */
               ef(P)   =   MIN(ef(P),ef(P_i));   af(P)   =
               MAX(af(P),af(P_i));
               P = P ∪ P_i; i = i + 1;

             end
          else SEARCH(f);
          P_i = P_i - f^n;

        end
   end
```

The knowledge derivation paths as the primary output of

this search are represented by a set of paths. Different paths in such a set, however, must have distinct sets of edges instead of distinct sets of vertices. As a consequence, two paths are considered to be the same path if their sequences of vertices on the two paths are mapped to identical ordered sets of edges. In the FG of Fig. 7-1, for example, the path from BEAM.WIDTH to SHAFT.WEIGHT and the path from BEAM.LENGTH to SHAFT.WEIGHT would turn out to be the same path which is composed of SE1 and SE2.

Procedures by Query Type

For each type of query, we describe a specific algorithm for identifying the relevant rules and constraints. The source or target parameters are not specified in queries of types II and IV unlike for query types I and III, so they have to be decided by the query processor.

1) Types I and III

For query types I and III, the source and target parameters are explicitly specified. The type III is in effect a repetitive version of the type I. This type requires a different query execution strategy. However, the search algorithm for identifying required rules is basically the same as that for type I.

S ← given source parameters;

T ← given target parameters;

Algorithm 7.3.2;

Consider for example a query of type I, "What would the COST of SHAFT be if the WIDTH of BEAM is doubled ?" For the source and target parameters, S = {BEAM.WIDTH} and T = {SHAFT.PRICE}, the paths P will be identified as a subgraph of nine edges, i.e., SE1, SE2, ME1 (with ME2 as the trigger), ME3, SE3, ME4, CE2, SE4 and SE5, in Fig. 7-2. The corresponding nine rules and constraints in the order specified by the subgraph are required to process the query with BEAM.WIDTH and SHAFT.PRICE as S and T, respectively.

2) Type II

For query type II, the target parameters are not given in the query. Hence, its target parameters are first identified. In fact, we can decide only the potential output parameters for the query, that is, they may turn out not to be involved on the final answer to the query. Specifically, the target parameters are all explicit head vertices that can be reached from the source vertices. Accordingly, the paths are all possible paths from the specified source entities to any of the identified target parameters. The final output parameters can be fixed only after comparing the current values of the potential output parameters with the values

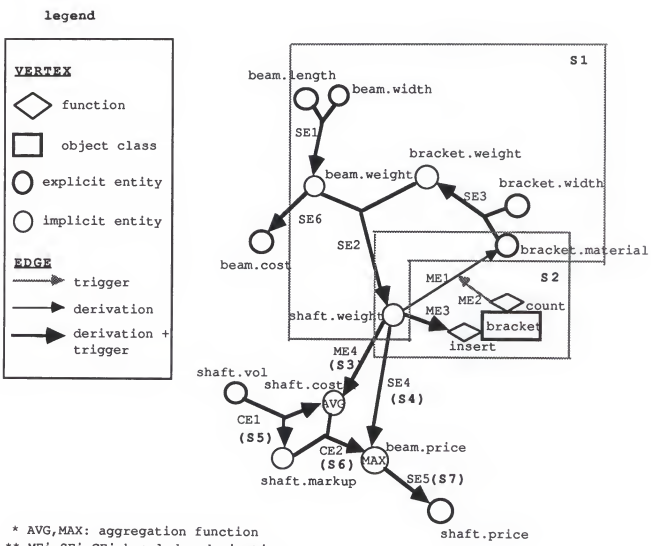


Fig. 7-2 A Function Graph for CSs: ME,SE,CE and IE

derived from actual query execution.

$S \leftarrow$ given source parameters, S_2;

$T \leftarrow$ all consequent vertices that are explicit, i.e., V = {v} = $\cup_i W_c(e_i)$ such that com(v) = EX;

Algorithm 7.3.2;

$T \leftarrow T \cap \cup_i W_c(e_i)$ where $e_i \in P$

Consider the FG shown in Fig. 7-3. A type-II query with BEAM.WIDTH, for example, as its source parameter will identify as its target parameters, three data items, BRACKET.WEIGHT via SE1, SHAFT.WEIGHT via SE1 and SE2 or ME2, and SHAFT.COST via SE1, SE2 or ME2, and SE3, and the output of search will be the graph of SE1, SE2, ME2 and SE3.

3) Type IV

For query type IV, the source parameters are not given in the query. Specifically, the source parameters are all explicit tail vertices such that there is at least a path from the source parameters to the given target vertices.

$S \leftarrow$ all antecedent vertices that are explicit, i.e., V = {v} = $\cup_i W_i(e_i)$ such that com(v) = EX;

$T \leftarrow$ given target parameters, T_4;

Algorithm 7.3.2;

$S \leftarrow S \cap \cup_i W_i(e_i)$ where $e_i \in P$;

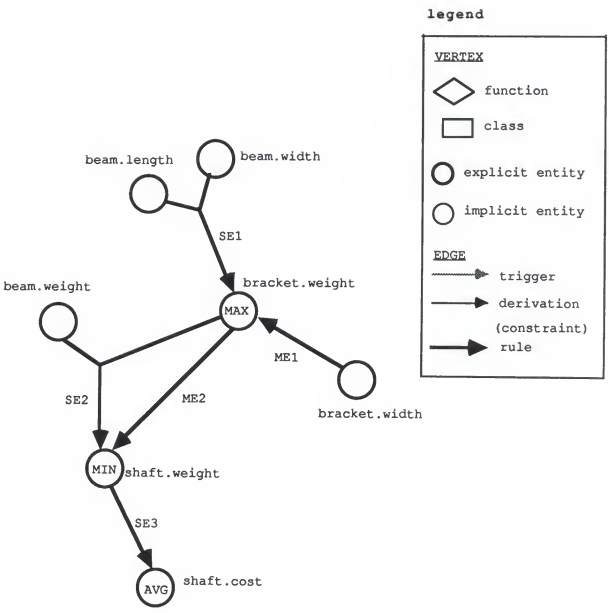


Fig. 7-3 Partitioning Graph into Segments

Similar to those in a type-II query, the source vertices are to be identified for a type-IV query. However, the searchfor paths in this type is directed backward, that is, from the target to source vertices. The search for a type-IV query with SHAFT.WEIGHT in Fig. 7-3, for example, as its target parameter starts at SHAFT.WEIGHT and finds BEAM.WEIGHT via SE2, BRACKET.WEIGHT via SE2 or ME2, BRACKET.WIDTH via ME2 and ME1, BEAM.WIDTH and BEAM.LENGTH via ME2 and SE1, as its source parameters. The output of search would be represented by a graph of edges, SE2, ME2, ME1 and SE1.

7.4 Decomposition of Query

In the previous step of query processing, the search for the required rules and constraints was carried out in a global scope. That is, we have not been concerned with which component systems the required rules and constraints are associated with. In general, different portions of the virtual global inferencing chains formed by such identified rules and constraints may belong to different component systems. Thus, such virtual chains must be decomposed into units of actual processing (called tasks) according to their association with particular component systems before they can be executed in proper component systems. Also, partial chains resulting from the decomposition need to be ordered such that their (partial)

results progressively derive a complete answer of the query as represented by the identified global chains. In this step, a query is decomposed into an ordered set of tasks, which collectively represent a query. The data items required for exercising these involved rules and constraints also need to be correspondingly identified.

7.4.1 Partitioning of Query into Tasks

A task is the basic unit of processing. That is, each task is expected to be processed by one of the component systems without interactions with the other component systems except for getting input data and giving output data for the rules on its boundary. For a task to be an independently-processable unit, the edges in the task must satisfy the following two conditions: 1) all the edges belong to the same component system, and 2) the preceding paths of each edge belong to the same sequence of tasks.

The necessity of Condition 1) is obvious in that each rule has its own underlying inferencing mechanism and interpretation scheme. To apply Condition 1) amounts to finding a maximum spanning graph composed of edges from the same component system. For example, a path of SE1, SE2 and CE2 in Fig. 7-1 are partitioned into two connected subgraphs (also called segments), i.e., one of edges SE1 and SE2, and the other of the edge CE2. In this particular case, the two

segments satisfy Condition 2) as well, so they directly correspond to two tasks for the query along with their input and output parameters (e.g., BEAM.LENGTH, BEAM.WIDTH and BRACKET.WEIGHT as the input parameters, and SHAFT.WEIGHT as the output parameter for the first task.)

Condition 2) stems from the fact that a task is defined to be a unit processable without data exchange with the external world except for its input parameter values and its output parameter values. This condition could be simplified to one that only the immediate preceding task be the same for every path in a task since the condition on the other preceding tasks can be satisfied recursively. To elaborate Condition 2), two facts are considered. First, a maximum spanning graph satisfying Condition 1) may have more than two terminal vertices. A terminal vertex refers to a vertex on the boundary of such a graph. It corresponds to either an input or output parameter of a task. If different starting vertices in such a graph are connected with different preceding segments, the graph has to be broken further down into smaller segments. The edges even with different ancestor tasks in a segment might appear to be executable as one "monolithic" task, if only the input parameters are provided in proper timing. Consider as an obvious counterexample where a graph satisfying Condition 1) spans both a preceding and trailing segment of another segment S_2. In the FG shown in Fig. 7-3, a path from BEAM.LENGTH to SHAFT.COST consists of SE1, SE2 or

ME2, and SE3. A subgraph with three edges SE1, SE2 and SE3 all from SE, satisfies Condition 1). However, the entire graph of SE1, SE2 and SE3 apparently cannot be invoked as a single task since some of its parameters depends on the parameters of the other segment, ME2, and vice versa. That is, the edge, ME2, is both a preceding edge and trailing edge in relation to the same component system, i.e., SE, so that they cannot be serialized. As a result, such a graph must be divided further into smaller segments with respect to the dependency with its adjacent segments. That is, SE3 is separated from the rest of the graph to form another segment by itself.

Second, a vertex may have associations with more than one component system. Such a vertex is either a simple connecting vertex with one incoming and one outgoing edges, or a complex vertex where more than two edges coincide. Though the edges, SE1, SE2 and SE3, of the same component system, SE, are connected to form a graph satisfying Condition 1) as shown in Fig. 7-3, the graph must be divided into two segments. The obvious reason is that the graph must interact with another system, CE, on an intermediate vertex, i.e., SHAFT.WEIGHT.

7.4.2 Ordering of Tasks

A global query has been partitioned into segments based on the basic criterion of whether a portion of the query is executable independent of the other portions. Unless the

partitioning for a query result in a single segment, the segments have to be properly ordered according to their interdependencies. In general, two expertises (parts of expertises, precisely) may be interrelated in one of the three patterns such as precedence, parallelism and no relation. These relationships can automatically be captured if they are linked according to their common input and output parameters.

To represent the dependency such as precedence and parallelism among segments (accordingly, tasks), we extend the conventional precedence graph [PET85]. A precedence graph is in essence a directed acyclic graph, in which a node represents an entity such as a statement or process. Especially, nodes with multiple incoming and outgoing branches are the focus of the concept [PET85]. However, the precedence graph algorithms cannot directly apply to our approach since our task scheduling may involve cycles. Our method to overcome this shortcoming is to treat an entire cycle as a single node. The trick is that a cyclic path as a whole is processed like a subroutine in a program before advancing to a next node in the precedence graph.

With the above extension, the precedence graph technique is applicable to our task scheduling. In the extended precedence graph, a node corresponds to a task. A group of tasks forming a cycle is treated as a single node. A directed edge represents the precedence relationship between two adjacent tasks.

7.4.3 Overall Algorithm for Query Decomposition

Specifically, the graph(s) of edges as output from the preceding step is vertex-partitioned into sub-graphs (or segments) such that the rules corresponding to each segment are autonomously processable within one component system. By the partitioning, such a graph would result in an ordered set of subgraphs, {G}. The order among subgraphs are determined by their interdependencies in terms of their common entities. Each subgraph corresponds to a task. A subgraph becomes a task when it is encapsulated with its input and output parameters and the id of its associated component system. The input and output parameters of a task refer to the parameters required to start the task and the parameters to be derived by the task, respectively.

A segment G is a maximal set of edges which satisfies the following two conditions:

1. The edges in a segment belong to the same inferencing mechanism. Specifically, $l(e_m) = l(e_n)$, if $e_m, e_n \in G$.
2. The edges in a segment are preceded by the same set of segments. Specifically, $G^i = G^j$, if $e_m, e_n \in G$, $G^i = \{e \mid e > e_m\}$ and $G^j = \{e \mid e > e_n\}$.

The input and output parameters of a task is decided as follows. This decision is basically based on whether data need to be exchanged on the entities on the boundary of the corresponding segment. First, if a boundary entity is one of the source or target parameters of a query, it becomes its input or output parameter, respectively. Second, if two or more tasks have an entity in common, it accordingly becomes an input or output parameter of each of the tasks. Such an entity corresponds to a confluent or divergent vertex in a Function Graph. The schemes for handling some special situations are as follows.

1) Mode of Chaining--If a task is to be assigned to a backward-chaining component system, additional input parameters may be required besides its input parameters, which are originally identified.

2) Cyclic Path--The notion of cyclic path would be lost from this step of query processing, where a cycle probably is broken down into several tasks. However, a unique id is assigned to each cycle so that it provides the following query execution step with a handle of the cycle.

3) Confluent Entity--A confluent entity belongs to as many tasks as the number of its intersecting paths. Each confluent

vertex is given a unique id for the reason similar to that for the cyclic path.

The overall partitioning algorithm is given below:

Algorithm 7.4.1 Partition identified rules into an ordered set of rule groups (and into corresponding tasks.)

INPUT: a set of sub-graphs $P = \{P_i | P_i \subseteq FG\}$ as identified in the preceding step.

OUTPUT: an ordered set of tasks, $K = \{K_i\}$.

METHOD: For every edge e_i emerging from the tail vertices of an edge e_c, check: 1) if its component system (cs) is the same as that of e_c, and 2) if each of edges ending at any of tail vertices, W_t, of e_i belongs to the preceding segments of e_c. If both 1) and 2) hold, expand the segment including e_c by adding e_i. Otherwise, create a new segment with e_i. Convert each segment $G = \{e_i\}$ to a task defined by a 4-tuple (cs,IN,OUT,G) where cs is the associated component system and IN and OUT are input and output parameter sets, respectively.

begin

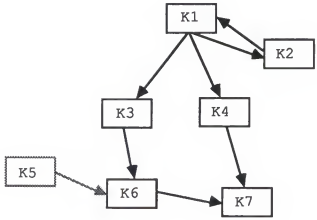
```
i=1; /* i = current task number */
for each Pj in P do
     for each edge e with no ancestor edges do
          begin
            IN(Ki) = IN(Ki) ∪ (Ws(e) ∪ Wt(e));
            SEARCH(e,i);
```

```
                    i=i+1;
                end
    end

    procedure SEARCH(e,i):
        begin
            for each vertex v in W_c(e) do
                for each edge f such that v ∈ W_i(f) do
                    begin
                        let e_ij ∈ G_i;
                        if l(e)=l(f) and W_i(f) ⊆ (∪_j W_c(e_ij)) then G_i
= G_i ∪ {f};
                        else begin
                            cs(K_i) = l(e);

                            /* getting output parameters for the
                            current task */
                            OUT(K_i) = OUT(K_i) ∪ W_c(e);
                            i=i+1;

                            /* getting input parameters for the
next task */
                            IN(K_i) = IN(K_i) ∪ (W_a(f) ∪ W_i(f));
                        end
                        SEARCH(f,i);
                    end
        end
```

By this partitioning on vertices, a Function Graph is simply delimited along "border" vertices into a link-disjoint, exhaustive set of segments, i.e., $S_i \cap S_j=\emptyset$, if $i\neq j$, and $\cup_i Si=P$, so the orders among the edges remain intact (note that adjacent segments overlap to the extent that they share vertices on their common boundary.) Also, the precedence relation among segments is uniquely determined because each segment is partitioned so that all the edges in a segment have the same preceding segments (Condition 2 above.) As a result, the Function Graph after the partitioning will result in an ordered set of segments. The corresponding tasks are defined

as a 4-tuple $K_i=(cs_i,IN_i,OUT_i,G_i)$, where cs_i denotes the associated component system; IN_i and OUT_i denote the input and output parameter sets, respectively; G_i denotes the segment. IN_i specifies data to be provided to start K_i, and OUT_i specifies data to be produced as the result of K_i. IN_i and OUT_i correspond to the vertices on the boundary of the task, K_i. Specifically, $IN_i = \{v \mid v \in (v_t \cup v_a)$ where v_t and v_a are on the boundary of segment G_i and $v_t \in W_t$ of any edge and $v_a \in W_a$ of any edge in segment $G_i\}$, and $OUT_i = \{v \mid v \in v_c$ where v_c are on the boundary of segment G_i and $v_c \in W_c$ of any edge in segment $G_i\}$. The execution order of the corresponding tasks are decided according to the relationships among their I/O parameters. Specifically, if two tasks K_i and K_j are related such that $IN(K_i) \cap OUT(K_j) \neq \emptyset$, K_i precedes K_j in execution order, e.g., S_4 (corresponding to K_4) precedes S_7 (corresponding to K_7) in Fig. 7-2.

For example, the path set P = {SE1, SE2, ME1, ME3, SE3, ME4, CE2, SE4 and SE5} will be partitioned into six segments, i.e., S_1, S_2, S_3, S_4, S_6 and S_7 in Fig. 7-4, where S_1 is composed of SE1, SE2 and SE3; S_2 of ME1 and ME3; S_3 of ME4; S_4 of SE4; S_6 of CE2; S_7 of SE5. (K_5 will be added in the subsequent step of query processing.) The corresponding six tasks, K_i, i = 1,2,3,4,6,7, are ordered as shown in Fig. 7-4. Each of the tasks becomes a node in the precedence graph. Two tasks K_3 and K_4, for example, are concurrently processable because no precedence relationship



TASKS:(cs,IN,OUT,set of involved edges)

K1=(SE,{beam.length,beam.width,bracket.width,
bracket.material},{shaft.weight},{SE1,SE2,SE3})
K2=(ME,{shaft.weight},{bracket.material},{ME3,ME1})
K3=(ME,{bracket.weight},{shaft.cost},{ME4})
K4=(SE,{shaft.weight},{beam.price},{SE4})
K5=(CE,{shaft.vol},{shaft.markup,shaft.cost},{CE1})
K6=(CE,{shaft.markup,shaft.cost},{beam.price},{CE2})
K7=(SE,{beam.price},{shaft.price},{SE5})

Fig. 7-4 Scheduling of Tasks Corresponding to Segments

exists between the two tasks. A cycle is indicated between $K_1$and K_2. Notice that K_5 is not included in the output of thisstep, but will be included for an initialization purpose as will be discussed in the local query translation step.

7.5 Execution of Query

A global query has been decomposed into an ordered set of tasks, each of which is (autonomously) executable by one of the component systems. Once the tasks are assigned to their associated component systems, they will be executed independently of each other. However, it is necessary to have some global coordination over their independently-working component systems. The objective of this step of query processing is to coordinate the execution of such an ordered set of tasks and to resolve conflicts among their outputs in order to produce an answer to the query.

The global functions in query execution basically are: 1) the assignment of the tasks to the corresponding component systems according to the identified order, 2) the compilation of the results of the tasks and the evaluation of these results for a global query optimization, and 3) the assembly of these answers into the global answer. The input to this step is the ordered set of tasks corresponding to a query and related information, and the output is the answer to the

query.

We shall first describe these global functions in terms of buffer management, evaluation of intermediate results, and communication with component systems. Based on these individual global functions, we develop the procedure for query execution for each query type.

7.5.1 Sizes of Buffer

Besides the usual buffering needs due to the disparity in processing speed among different component systems, we have to consider some additional buffering requirements unique to our system. Specifically, we discuss the appropriate amounts of buffer storage to be allocated for several cases: (1) vertices on the boundary of two adjacent tasks, (2) confluent vertices associated with multiple component systems, (4) iterative query and (4) cyclic path.

An entity on the boundary of two adjacent tasks needs to buffer its value resulted from the execution of preceding task until it is shipped out as an input for the trailing task execution. Specifically, for T_1 and T_2 such that $cs(T_1) \neq cs(T_2)$ and $W = OUT(T_1) \cap IN(T_2) \neq \emptyset$, a unit of buffer is allocated to store the value of each $v \in W$.

At a confluent vertex, conflicts may occur among the competing values on the corresponding entity which are derived via its multiple incoming edges. The required size of buffer

for a confluent vertex v_c with the N incoming edges B_c = (N + 1) × unit buffer.

As a result, the total buffer size required for a single-session query $B_s = 1 \times N_j + B_c \times N_c$ where N_j denotes # of junction vertices involved in the query and N_c denotes # of confluent vertices. For a multi-session query, each session may include the above two kinds of vertices. Also, additional storage for the evaluations and comparisons of intermediate values over many sessions to detect a trend in those values. The required buffer size for M-session query B_M for N_t target condition variables is $M \times N_t + B_s$.

For a cyclic path, a series of values on the reference entity are analyzed to detect any trend. In effect, multiple sessions are performed over the cyclic path. The required amount of buffer B_{cy} for a cycle with a preset number N_{cy} of iterations, $B_{cy} = N_{cy} \times$ unit buffer size + B_{sl} where B_{sl} denotes the buffer size required by a (partial) single-session consisting only of segments involved in processing the cycle.

7.5.2 Evaluation of Intermediate Results

Tasks assigned to their associated component systems are executed with no knowledge of the existence of the other component systems. Thus, it is the responsibility of the global query processor to resolve the conflicts that may occur among the values derived by different component systems on the

same entity. Specifically, the global query processor deals with three kinds of conflicts such as confluent edges, cyclic paths and repetitive queries. Note that we are not concerned with conventional issues such as resolutions between incompatible variables [GOL83].

1) Confluent Vertex: Some special query optimization is performed to resolve conflicts among many values on a confluent vertex, each derived via one of the parallel paths [AHO74] (by its associated component systems.) This conflict resolution is performed automatically by the query processor or interactively by the query issuer. The automatic mode is based on an aggregation function as specified on each confluent vertex. Possible options for the automatic mode are three aggregations functions, AVG, MAX and MIN. They indicate that the maximum, minimal or average of such competing values is the optimal value on the entity. The optimal value of SHAFT.COST, for example, would be $v = AVG(v_3, v_5)$, where v_3 and v_5 denote the values SHAFT.COST derived for K_3 by ME and for K_5 by CE, respectively. With the user intervention mode UND specified, the user is provided with a chance to choose the best value.

2) Cyclic Path: The handling of a cycle is based on the trend that the results of its repetitive execution would show. For a cycle $p = ((e^c_i), V_R)$, an equilibrium is sought in the

sequence of values for V_R after N iterative executions of a cycle. There are three possible trends in such a sequence of values: convergent, divergent and invariant. Specifically, it is determined to be convergent to v_i if $|v_i - v_{i+1}| < \Delta$ for $i \geq N$, where v_i denotes the value of V_R from i-th iteration, Δ denotes a tolerance for judging the "stableness" of the values of V_R, and N denotes the number of iterations performed. A trend of no changes in such values, i.e., $v_i - v_{i+1} = 0$, is a special type of equilibrium [BUC85]. With an equilibrium found, the value for V_R in the final session is determined as its value. If no equilibrium can be reached, the cycle is concluded to be nondeterministic. In an invariant trend, the constant value would be taken as its final value.

3) Repetitive Query: In case the target condition given in a type-III query cannot be satisfied in one inferencing chain (corresponding to one execution of the ordered set of tasks), repetitive processing would be required until the condition is met. In each inferencing chain (or session) of such a repetitive processing, the source conditions are modified in a direction to reduce the deviation of the values of target parameters derived in previous sessions from their values given in the target condition.

To modify the source condition, an increment to be added to the value of the source parameter is adjusted according to the results of the preceding sessions. The adjustments are

prescribed by the direction and magnitude of the deviation of the values on the target parameters in the current session from those in the preceding sessions. If the direction of deviation has been reversed, the adjustment of the source parameter in the previous session is determined as overdone. Then, the size of increment is halved and negated for the next session. Suppose, by applying an increment δ to the source parameter s_2, we have obtained a series of values on a target parameter, $v_1 > v_2 > v > v_3$, where v_i = values from i-th session, v = target value. Then, $-\delta/2$ should be a plausible increment to be added to s_2 in the next session. On the other hand, when the deviation shows a monotonically decreasing pattern, the increment is kept over the subsequent sessions until any disruption in the pattern occurs.

However, we would encounter various nondeterministic patterns in the deviation. Thus, user intervention mode is an inevitable provision against such situations. This mode allows the user to step in to adjust the source conditions. In this mode, a type-III query is in effect equivalent to a series of type-I queries with a facility for the user to modify the source conditions between sessions.

7.5.3 Communications with Component Systems

Besides the general issues in conventional distributed system, we need to consider: assigning tasks, and collecting

the results of tasks.

1) Assigning Tasks--Tasks are assigned to their associated component systems with data on their input and output parameters and other required information. The input parameter values for each task need to be provided in a predefined order.

Two methods of communication are possible to provide input parameter values in assigning tasks to their associated component systems. One method is incremental communication where the information needed is incrementally provided on demand. The second method provides all information at one time, i.e., when a task is assigned. This method is applicable only when all the required parameter values are known a priori. Otherwise, the first method will be the only choice though it may be an inefficient one. The second method is more efficient than the first one in that the actually needed intermediate input parameters may be fewer than the identified parameters IN_i for tasks other than the starting tasks such that $IN_i \cap S \neq \emptyset$, depending on the current results of inferencing. Note that all the initial values to start a component system need to be provided at the beginning of a session. We used this method in building our prototype system.

2) Collecting Results of Tasks--Each task is carried out in one of the component systems with no knowledge of the other

tasks. As a consequence, their results tend to be scrambled in order, due to differing processing speeds of systems. As discussed earlier, individual tasks need to be marked with its task number and session number so as to restore, if needed, the correct order at subsequent stages of transaction. Also, some buffering facilities are provided in this respect.

The identified paths reflect only possible knowledge derivation processes, so that their outputs will depend on the actual results from their underlying inferencing process. Depending on its intermediate results, an inferencing process may terminate before all the output parameters identified are derived. To detect such situations, we can use a timer as in general distributed systems [TANE88].

7.5.4 Query Execution Mechanism by Type of Query

Depending on its type, one or more sessions are established for a query. A session denotes one execution of the ordered set of tasks corresponding to a query as scheduled in the task scheduling step. Multi-session is needed for type-III queries. To process a type-III query, the source parameter values are adjusted according to the outputs of its preceding sessions. Regardless of the query type, a cyclic path needs to go through multiple sessions until a pattern such as no change, convergent or divergent [RAM88] is detected in the series of values generated for a reference vertex of the

cyclic path. The primary difference from processing a type-III query is that the parameter values are not adjusted in each session. First, we outline the general procedure of session. Then, we describe the procedure for each type of query.

Processing of Session: The following algorithm is basically to successively activate ordered tasks using the parameter values derived by their preceding tasks until obtaining values on the target parameters. To start a session, the adequate amount of storage is secured to buffer intermediate results returned from component systems executing tasks. Then, the tasks at the top of the schedule are assigned to corresponding component systems. In assigning tasks, it is checked if there exists any independency among tasks such that they are processable in a concurrent fashion. As soon as their outputs are returned from the component systems, they are evaluated to resolve various conflicts. Some special query optimization is performed to resolve conflicts among competing values on a confluent vertex, each derived via one of the parallel paths ending at the vertex [AHO74] (by its associated component systems.) If the resolved outputs constitute valid inputs to the trailing tasks on the schedule, they are in turn assigned to corresponding component systems. This procedure repeats until preceding tasks no longer derive any valid inputs to the trailing tasks. At this point, the outputs of the last tasks

are sent to the user as the final output after being properly formatted, if necessary. With the outputs of tasks, the global query processor deals with three kinds of conflict resolution such as confluent edges, cyclic paths and recurrent queries.

Algorithm 7.5.1 Perform a Single Session.

INPUT: an ordered set of tasks {K} corresponding to a query, and its source and target parameters

OUTPUT: the values on target parameters of the query

begin

Assign task id = (session #, task #) to each task;
Secure required amounts of buffer;
Assign the "starting" tasks $K_s=(cs_s, IN_s, OUT_s, G_s)$ such that $IN_s \cap S \neq \emptyset$ to cs_s.

/* some cs_s may request "initial" data v_i for their initialization. */
Provide cs_s with "coherent" data on IN_s and v_i.

/* coherent data means the instances forming an aggregation hierarchy in DB */
while $\cup_i OUT_i \not\supseteq T$ for all tasks K_i that have already executed, and there exists K_j such that $W \subseteq (\cup_i OUT_i)$, where $W = \{v \mid v \in IN_j$ and mode(v) = EX$\}$ do
    begin

        Store the values of OUT_i returned from cs_i.
        if there exist competing values v_i, $1 \le i \le n$, on any $e \in E_c = OUT_{i1} \cap OUTi_2 \cap \ldots OUT_{in}$ then
            for each e in E_c do
                Apply aggr(e) to v_i to calculate an optimal value on e.

        Assign K_j to cs_j with the values of OUT_i as the values of IN_j.
    end
return the values of OUT_i such that $OUT_i \cap T$.

end

Consider, for example, the execution of a session composed of the ordered set of tasks as shown Fig. 7-4. Let's ignore K_2 to postpone the discussion on the cycle until the following subsection. After assigning a session id, K_1 is executed by SE with its four input parameter values. The value of its output parameter, i.e., SHAFT.WEIGHT, is provided for ME to start K_3 if it is returned within a time limit. The value of the output parameter of K_3 derived from ME is in turn given to CE for performing K_6. (Note that the input parameter of K_6 is an implicit entity with respect to CE so the path needs to be extended backward until it starts with an edge with only explicit entities. We detail this issue later.) The two values of BEAM.PRICE derived by SE and SE (as the output of K_4 and K_6) are resolved according to the option attached to BEAM.PRICE. The larger of the two values is fed back to SE to derive the value of the final parameter through K_7.

Processing of Cycle: To detect a trend in the values on a reference vertex of the path, a cycle is executed a number of times over the cyclic path. Each cycle constitutes a session. The output values from a session are provided as the values of the input parameters of the tasks in the subsequent session. By applying the algorithm described before, the global query processor determines the trend in the series of values. If the trend is judged to be convergent, the execution

of the cyclic path is repeated up to some additional number of times in an attempt to reach an equilibrium on the resulting values within a tolerance. If divergent, the cycle would be concluded to be non-deterministic to cause user intervention or termination of query processing. If constant, we simply stop repeating the cycle and exit from the cycle with the current values as the final result from the cyclic path. Notice that, even though the execution of a cycle also goes through repetitive processing, the evolving results are merely observed without modifications of the original source condition in the query unlike for a type-III query.

Algorithm 7.5.2 Processing of Cycle

INPUT: a set of tasks, $\{K_i \mid K_i = (cs_i, IN_i, OUT_i)\}$, in a cycle, $P_c = (\{e^c_i\}, V_R)$.

OUTPUT: a final value on V_R, the reference vertex of the cycle.

```
begin
     IN0 ← V_R
     for i ← 1 until INIT_ITERATION do
        begin
          IN_i = v_i;
          v_i = Procedure 7.2 /* execution of one session
          */
          V ← V ∪ v_i;
        end;

     if |v_i - v_{i+1}| > Δ for i ≥ INIT_ITERATIONS
          then inform the user of its divergency;      */
          Δ = the tolerance */
          else begin
               while the value is not stable, i.e., | v_i
```

```
                    - v_{i+1} ¦ > Δ, and the number of executed
                    sessions does not exceed a preset number,
                    session # < MAX_ITERATION
                       do
                         i ← i + 1;
                         IN_i = v_i;
                         v_i = Algorithm 7.5.1 /* execution of
                         one session */
                       end
            end
      v_c ← v_i ;

end
```

Following are the algorithms for specific types of queries.

Type I

The type-I query is executed in one session unless a cycle is involved. The overall procedures are in the order of storage allocation, assignment of tasks, compiling and analysis of intermediate results of tasks, decisions on final results, and formatting and displaying of final outputs.

```
INPUT: a set of tasks, {K_i¦ K_i = (cs_i, IN_i, OUT_i) ),
corresponding to a query, Q = (TYPE-I, S, T, OPTION)
OUTPUT: answer for the query, Q.
begin
     for each task K_i ∈ {K_i} do
        begin

          NEW (the new values of OUT_i) ← Algorithm 7.5.1;
          If OPTION = RELATIVE
               then begin
```

```
                CURRENT ← the current value of OUT_i
                retrieved from the underlying
                databases.
                for each v^n_i ∈ NEW and each v^c_i ∈
                CURRENT and v^n_i = v^c_i do
                OUTPUT ← v^n_i / v^c_i;

            end
            else        OUTPUT ← v^n_i ;

        end
```

Type II

The procedures are basically the same as those for type I. The difference is that, among the identified target parameters, only those whose values have been changed are returned in the query answer.

```
INPUT: a set of tasks, {K_i | K_i = (cs_i, IN_i, OUT_i, G_i) },
corresponding to a query, Q = (TYPE-II, S, T)
OUTPUT: answer for the query, Q.
begin
    for each task K_i ∈ {K_i} do
        begin
            NEW (the new values of OUT_i) ← Algorithm
            7.5.1;
            CURRENT ← the current value of OUT_i retrieved
            from the underlying databases.

            for each v^n_i ∈ NEW and v^c_i ∈ CURRENT do
                if v^n_i ≠ v^c_i then OUTPUT ← v^n_i;

        end
```

Type III

A type-III query requires multi-session to satisfy the given target conditions. Each iteration constitutes a session. Each time a session finishes, the results are compiled and compared with those in the preceding sessions. Based on the comparison, a new source condition for the next session is calculated in an attempt to meet the target conditions. With the AUTO mode specified in a query, the source conditions are adjusted without user interventions according to the algorithm described before. The query execution terminates when all the conditions have been satisfied, or it is aborted by the user.

For a query, (TYPE-III, S, T, AUTO, Δ), the initial session starts with the original source parameter values, ($s_i = v_s$). The output values on T are compared with their values v_t given in the query to check if the target condition are satisfied within the tolerance, Δ. Unless the target conditions are satisfied, the value of s_i would be replaced by a new value $v_i = v_s + \delta$, where δ is an increment as decided by the algorithm. With the new value, another session is performed as in the previous session. This iteration repeats until the target condition is satisfied or up to a preset maximum number of sessions.

INPUT: a set of tasks, $\{K_i | \ K_i = (cs_i, IN_i, OUT_i)\ \}$, corresponding to a query, Q = (TYPE-III, {s [= v_s]), {t

= v_t), OPTION, [Δ])

OUTPUT: answer for the query, Q.

```
begin

    if v_s is given
       then INi ← v_s
       else IN_i ← values of v_s from the underlying
    databases

    while the condition is not satisfied, | v_t - v | >
    Δ, and the number of executed sessions does not
    exceed a preset number, session # < MAX_SESSION do
       begin

         session # = session # + 1;
         for each task K_i ∈ {K_i} do
            begin
              for each v_i IN_i ∈ S do
              v_i ← v_s + δ; /* according to the scheme as
              given above */

              NEW (new values of OUT_i) ← Algorithm
              7.5.1;
              v ← v^n_i ∈ NEW;
            end
            OUTPUT ← v^n_i ;

       end

end
```

Type IV

In this step, the type-IV query reduces to mere retrieval operations for the identified source parameters. The current value is retrieved as possible for each of the output parameters. However, values are retrievable only for explicit vertices, so only the entity names are output for the source parameters corresponding to implicit vertices.

INPUT: a set of tasks, $\{K_i \mid K_i = (cs_i, OUT_i)\}$, corresponding to a query, Q = (TYPE-IV, T)

OUTPUT: answer for the query, Q.

for each task $K_i \in \{K_i\}$

OUTPUT ← the values of OUT_i retrieved from underlying databases;

7.6 Processing of Tasks in Component Systems

Once tasks are assigned to their associated component systems, the global query processor simply waits until their outputs are returned from the component systems. Thus, the component systems are responsible for returning the outputs of tasks to the global query processor.

Specifically, the query processing in a component system goes through the following steps:

(1) Tasks that are assigned to the same component system and belong to the same session are merged into one task.
(2) Each task resulting from such mergence is translated into a query to its associate component system. For this translation, we need some mapping information to be used to translate a merged task as a whole into a query for the associated component system.

(3) The values of the initial entities for the query are obtained from the underlying DB or remote sites (or DBs underlying other component systems) via the global query processor.

(4) The query execution continues until all of the output parameters of its corresponding task are obtained or its execution can no longer proceed.

(5) The query outputs are stored in a sharable file, from which they are forwarded to the global query processor after proper formatting.

For tasks to be performed in their associated component systems, they need to be transformed into executable forms for those systems. Such a transformation for rule-based systems in general, is more difficult than query translation in distributed DBs. This is due to the unique characteristics of the knowledge derivation using rules. First, rules are interdependent so that it requires an orderly execution to obtain data by exercising rules, while data items in conventional databases are mutually independent allowing a random access to each data item. Second, general rule-based systems unlike DBMSs are not equipped with a (formal) query language. Thus, we develop a different query translation scheme for general component systems with arbitrary user interfaces. In this respect, we consider two properties of rule-based systems in general: 1) inferencing in a rule-based

system can start only at a designated way, and 2) the queries in conventional rule-based systems are formulated in ad hoc forms, e.g., in an interactive manner.

As the possible component systems of our integrated system, we consider three major types of rule-based systems: constraint-management systems, deductive DB systems and general expert systems. According to Property 1) stated above, a proper initialization of inferencing is required no matter what portion of the inferencing chain the task corresponds to. All tasks in the same session and associated with the same initialization procedure can then be performed as parts of the same inferencing chain. However, tasks contain no specific information on such query format. In this respect, the FG provides only the "conceptual" information, that is, which entities are the starting and ending points of inferencing for each component system. Specifically, an edge whose tail vertices all are explicit vertices represents a possible starting point of inferencing in its associated component system. Such an edge in a task needs to be extrapolated into an initialization point in the component system to translate the task into a proper local query to the component system. By an initialization point of a system, we mean its actual starting point of inferencing, where its inferencing can start if all "initial" vertices are instantiated.

For such an extrapolation, each Local Knowledge Administration Module (LKAM) maintains a mapping table to

specify the mapping from conceptual starting and ending points of inferencing to the corresponding initialization procedure (or the input protocol) and the output protocol. For each local component system, the mapping table specifies which rules can be fired by, and what data are required to be instantiated in, each initializing procedure. Each of its entries represents those protocols for an explicit-tail edge.

7.6.1 Representation of Entry Protocols

For each initialization point of inferencing on a component system, we need to specify the entities to be instantiated to start inferencing. In general, however, a component system may have one or more initialization points of inferencing. Thus, we need to specify the path that leads to one of the initialization points in addition to the entities required for initialization. Consider, for example, a system that is based on a hierarchical control structure with a menu-driven user interface [PRA87]. Using that system, we will select from a menu at each level of hierarchy before we can get to an initialization point. For such a system, the first part of an entry protocol specifies a sequence of symbols, one symbol leading to each level in the user interface. Obviously, the first part of input protocol would become trivial for a system with only one initialization point. The second part of input protocol lists all the initial

vertices, which are to be instantiated to initialize an inferencing process.

For all practical purposes, the first part can be represented by a sequence of symbols. The second part is always a set of explicit vertices. Then, the complete input protocol for an explicit vertex v can be represented as a pair (S, V), where S is a sequence of symbols and V is a vertex set. Specifically, an entry of the table MT_m for component system CS is a tuple (m, S, V, E), where m=id of initialization procedure of CS; S = a sequence of symbols leading to the initialization point; V = a sequence of initial variables; E is a set of edges that can be fired directly from m, consequently, E = {e| for every $v \in W_1(e)$ mode(v)=EX }. In MYCIN, for example, the table would have only one tuple with V=(name,age,sex,race), and E={RULE 092 with the top goal REGIMEN} [BUC85].

7.6.2 Mapping of Tasks into a Query

In constraint management systems, each constraint has its own (explicit) triggering condition (e.g., an update operation on some entity) so that it can be triggered by itself. Thus, each task would directly be transformed into an update operation on the input parameters of the task and subsequent retrieval operations on its output parameters.

For deductive DBs, queries will be issued in a similar

manner as for conventional DBs. In a DB system based on Prolog, for example, the task K_7 = (SE, {BEAM.PRICE}, {SHAFT.PRICE}, {SE5}) would be transformed into a query, price(shaft, x)?, following an insert operation for a ground predicate, for example, price(beam, 100).

In the following, we identify and develop solutions to problems, which are unique in performing tasks in general expert systems, in the context of two major issues: mapping of tasks into local queries, and exchange of data (on I/O parameters of tasks) among component systems.

1) Handling of Implicit Entities in Each Task

The value of an implicit entity cannot be simply fetched from storage, but must be derived in its associated component system by exercising a chain of rules. Thus, if any starting vertex of a path is an implicit vertex, we need to track backward from the vertex in search of the nearest edge with only explicit vertices on its tail.

Algorithm 7.6.1 Handling of Implicit Entities

```
begin
    for each v ∈ IN such that com(v) = IM do
        while there exists e >> e_c and cs(e) = cs(e_c)
        and e is an edge such that com(v) = EX for each
        v ∈ W_t(e) do
            begin
                S ← S ∪ e;
                e_c = e;
            end
```

end

Suppose, for example, that a path traverses two vertices, BEAM.WEIGHT and SHAFT.WEIGHT, via the edge, SE2, in Fig. 7-1. Then, BRACKET.WEIGHT must also be instantiated to fire SE2. Since BRACKET.WEIGHT is an implicit vertex, a backtracking would also be required to find an edge with only explicit vertices, BRACKET.WIDTH and BRACKET.MATERIAL in this case.

2) Merging of Tasks

If more than one task in the same session have been identified to begin with the same initialization point of a component system, those tasks have to be performed within one inferencing process. That is to say, such tasks are to be merged into one task, which corresponds to a query for the component system.

Specifically, if tasks $\{K_i | K_i = (cs, IN_i, OUT_i, G_i)\}$ are "consequence" of the same initialization procedure, e.g., $\{K_5, K_6\}$ in Fig. 7-4, they are merged into a single segment K_m. Consequently, K_m would be translated into a single local query to their common component system cs. However, its I/O parameters may be fewer than unions of individual IN_i and OUT_i, that is, $IN_m = (\cup_i IN_i) - ((\cup_i IN_i) \cap (\cup_i OUT_i))$, e.g., SHAFT.VOL and SHAFT.COST, and $OUT_m = (\cup_i OUT_i) - ((\cup_i IN_i) \cap (\cup_i OUT_i))$, e.g., BEAM.PRICE. Note, however, that those parameters involved in interaction with other component systems are not

canceled out, e.g., SHAFT.COST between K_5 and K_6 for CE's interactions with ME with respect to K_3.

Algorithm 7.6.2 Merging of Tasks

Method: 1. For each task in the set of tasks assigned to a component system, find a minimum number of initialization points such that all of the input parameters of a task are among the data items that are to be initialized by the identified sets of initialization points.

2. If more than one task share the same initialization point, merge those tasks into one query so that such tasks are processed as one unit.

INPUT: a set of tasks, (K_i, $cs(K_j) = cs(K_l)$, where $j \neq l$)

OUTPUT: a set of tasks (K^m_n)

begin

for each K_j do
find a set of initialization points, $M_j = (m_i |$ $m_i = (S_i, V_i, E_i)$, initialization point i) such that $\cup_i V_i \supseteq IN_j$;

for each distinct M_j do /* tasks associated with M_j is denoted by K_{ji} */
begin

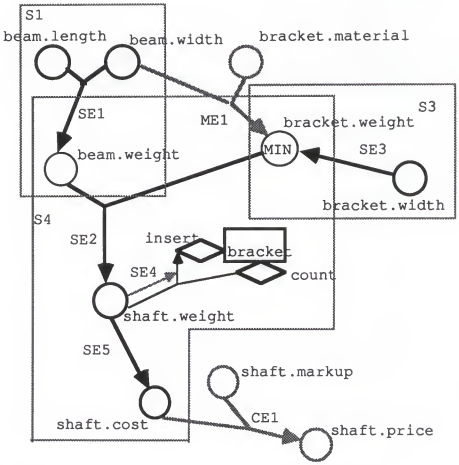
$IN_{mj} = (\cup_i IN_{ji}) - ((\cup_i IN_{ji}) \cap (\cup_i OUT_{ji}))$
$OUT^m_j = (\cup_i OUT_{ji}) - ((\cup_i IN_{ji}) \cap (\cup_i OUT_{ji}))$
$G^m_j \leftarrow G^m_j \cup \cup_i G_{ji}$
end

end

The two tasks, K_1 and K_4, in the Fig. 7-5, for example, will be combined into one task with two input parameters, BEAM.LENGTH and BEAM.WIDTH, via the common entity, i.e., BEAM.WEIGHT (an output parameter of K_1 and input parameter of K_4.) The common vertex BEAM.WEIGHT is cancelled out so it is no longer an input or an output parameter. Unless K_3 and K_1 are translated to different initialization points of the component system, SE, all of the three tasks K_1, K_3 and K_4 will be translated to one query with three initial entities, i.e., BRACKET.WIDTH, and the above two starting entities of K_1 (and K_4.)

3) Translation of Task into Query

Each merged task is translated into a local query based on the above mapping table. Specifically, we first search the table for an initialization procedure (m, V, E) such that all input parameters of a merged segment K_m are reachable from E. For this search, we can exploit Algorithm 7.3.1 with $S = \cup_i W_a(e_i)$ and $T = \{v_m \mid v_m \in IN_m$ and $mode(v_m)=IM\}$ (but, S and T swapped for backward-chaining systems), and select a path, P, such that $ef(P) = MIN(ef(P_i))$, and $mode(e_{pi1})=EX$, where e_{pi1} denote the starting edges of P_i. For example, S6 in Fig. 7-3 would be extended to include CE1 (the closest edge with only explicit entities, i.e., SHAFT.VOL) unless CE1 is already in the segment. In fact, most simple expert systems that have only one entry point would not need this search. Finally, a



TASKS assigned : (cs, IN, OUT, set of involved edges)

K1=(SE,{beam.length,beam.width},{beam.weight},{SE1})
K2=(ME,{beam.wdth,bracket.material},{bracket.weight},{ME1})
K3=(SE,{bracket.width},{bracket.weight},{SE3})
K4=(SE,{beam.weight,bracket.weight},{shaft.cost},{SE2,SE4,SE5})
K5=(CE,{shaft.cost,shaft.markup},{shaft.price},{CE1})
Km=K1+K3+K4=(SE,{beam.length,beam.width,bracket.width,bracket.weight},
{bracket.weight,shaft.cost},{SE1,SE2,SE3,SE4,SE5})

Fig.7-5 Mergence of Tasks in SE

merged task K_m is mapped into a sequence of initial variables V.

7.6.3 Data Exchange with the Global Query Processor

For exchanging data with existing expert systems, we need to consider: 1) interfacing problems such as data exchange on implicit entities and redirection of I/O devices, and 2) communication problems such as non-determinism in inferencing chain, intermediate parameters required in the course of inferencing, and premature termination of inferencing.

The data items through which its input and output can occur and its input and output devices for exchanging data with the external world are determined by the particular system structure of each component system. Consequently, the accessibility to some (intermediate) entities in an existing system are restricted according to its user interface. To achieve a tight integration as proposed, however, our approach should allow data exchanges to occur on any intermediate entity.

A junction vertex on the FG, which two or more component systems are associated with, reflects possibility of data exchange on the corresponding entity among those associated component systems. The execution of tasks with such a vertex as their common output parameter has to be synchronized, that is, competing values on their common parameter from different

tasks must be collected to be globally resolved before the subsequent tasks can start. Thus, component systems with adjacent tasks need to exchange (via the global query processor) their data on their common I/O parameters. If such a parameter happens to be an implicit vertex (i.e., some of the component systems may not be designed to exchange data on the entity identified as an I/O parameter), however, the corresponding data item, as it is, is inaccessible from or unprovidable by, external world. For such an entity, we need some provision by which data exchange can happen among the associated component systems. Specifically, if there exist tasks K_i and K_j with respect to some entity e such that mode(e) = IM, $e \in IN_i$, $e \in OUT_j$, and $cs(K_i) \neq cs(K_j)$, e.g., K_4 and K_6 with BEAM.PRICE, e must be converted to an explicit one. With the provision, such a vertex is in effect converted to an explicit vertex. In this conversion, however, we should guarantee: 1) no disturbance on the original inferencing mechanism of component systems and 2) no structural change on component systems.

One way of extending such accessibility is to modify user interface to multiply accessible entities. However, that method might not be feasible if it requires some major modification of user interface. Notice that the rules in rule-based systems by definition are modifiable. Exploiting this inherent nature, we find more acceptable methods as detailed below.

We implement this conversion for an implicit entity e, by replacing MODIFY(e) with two I/O primitives, i.e., WRITE(d_{out},e) followed by READ(d_{in},e) where d_{out} denotes an output device, e.g., screen, and d_{in} denotes an input device, e.g., keyboard [BRO85]. The effect of this replacement is that, if a value of e is derived, it is output to a sharable device d_{out} instead of the working memory, and an external value for e can be read through d_{in} into the working memory. The requirement 1) is met in that the effect of this conversion is confined to exchange of values via redirections of I/O (with some time delay.) The requirement 2) is met since the above example does not need to change the structural relationship of the component systems. The only change is made in their rule bases. However, the modifiability of the rule base is an inherent nature of rule-based systems [BUC85, BRO85, NII86 and JAC86].

7.6.4 Consideration of Miscellaneous Issues

We have so far dealt with inferencing that occurs in the orders specified in Function Graph. The actual firing sequences within each component system may be different from those identified via search of the Function Graph. Also, due to the conflict resolution within a component system, the data item corresponding to a confluent vertex may be derived more than once at different times. Such an anomaly can be handled

with a minor modification of communication protocols between the global query processor and component systems. For eaxmple, we may choose the value that is derived for the last time as the final value of such a vertex.

The remaining issues on the communications between the global query processor and component systems are not discussed in detail here, but we should point out that they can be solved with the above conversion and existing techniques in general distributed systems. For redirection of I/O devices, we can also utilize available facilities on operating systems [KERN84]. Also, premature termination of inferencing is a kind of abnormal termination in general communication.

CHAPTER VIII

PROTOTYPING OF THE APPROACH

We built a prototype system for integrating heterogeneous rule-based systems based on our approach. The system is written in C language and running on a SUN workstation. Its component systems are three independently-developed expert systems with different inferencing mechanisms and rule formats.

The main objectives of this prototyping effort are: 1) a qualitative evaluation of our approach by comparing the integrated global solution by our prototype system with the "patched" global solution from combining the independent partial answers by individual systems, when given the same queries, 2) a verification of the algorithms and schemes we have developed, and 3) an investigation on issues unique to our approach by simplifying the conventional issues in general distributed systems such as communication problem and message translation.

The overall architecture of this prototype system is shown in Fig. 8-1. In the Dynamic Information module of the Global Knowledge Processor (GKP), a tabular representation

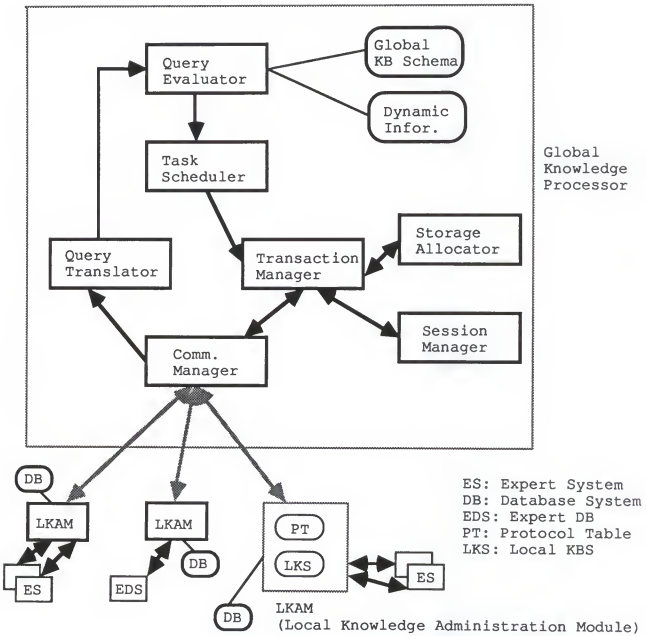


Fig. 8-1 System Architecture of the Heterogeneous Expert System

[AHO74] of directed graph was extended to accommodate a multi-vertex multi-graph (the Function Graph); In Query Evaluator module, Algorithm 7.3.2 was implemented by modifying a depth-first search algorithm for the general graph, so that cyclic paths and all possible paths rather than only the best path can be found [AHO74]; In Task Scheduler
module, Algorithm 7.4.1 for partitioning and ordering the identified edges was implemented, and the conventional acyclic precedence graph [PET85] was extended to account for cycles across tasks; The Transaction Manager module cooperates with Session Manager, Storage Allocator, and Communication Manager to execute global queries. The Transaction Manager calls Storage Allocator to secure storage proportional to the in- and out-degree of each vertex to buffer intermediate values on each corresponding I/O parameter, and then calls the Session Manager to perform Algorithm 7.5.1. The parameter values required by component systems are incrementally provided on demand.

In a separate development effort, three independent rule-based expert systems have been developed for use as the component systems of the integrated system. The three systems are based on the example expert systems described earlier. They are all written in C, and have been developed on IBM PCs.

These expert systems are limited in that the IF or THEN part of each rule can have only up to two clauses. However, they are more powerful than conventional expert systems

[SHO76, MCD82, BUC85 and BRO85] to the extent that their rule formats allow general functions to be embedded. Each system has only one initialization procedure like most existing expert systems, requiring no mapping table for initialization.

To focus on such issues that are unique to our approach, we simplified conventional issues in general distributed systems. For example, the communication problem between the GKP and the expert systems have been avoided by porting those component systems on the SUN workstation, where GKP is running. Specifically, the component expert systems and the global query processor run as four concurrent processes exchanging data via pipe facilities [KERN84] on UNIX. Each of the four processes displays its outputs on its own window so that we can visually check every point in the global query processing. Also, the usual lexical problems in query translation have been simplified by using common I/O protocols. Specifically, the expert systems were designed to exchange parameters in a strict form, $PARA_i$: val, instead of the usual natural-language messages.

Through this implementation, we proved that the knowledge representation part is sufficient for the query processor to identify proper component expert systems and decide proper execution orders among them. Our query processing part has been validated by obtaining the expected results from parallel and cyclic knowledge derivation paths as well as sequential paths. To verify the various schemes and algorithms in these

two (functionally-indivisible) parts of our approach, we checked several important issues such as data exchange among independent component systems, parallelism in knowledge derivation paths, query mapping for component systems with arbitrary user interfaces, and cycles across many component systems.

First, the adequacy of our scheme for data exchange was confirmed in terms of timing for synchronizing of independent component systems along a variety of knowledge derivation paths. In this respect, we observed that the three arbitrarily-interrelated rule-based systems running concurrently except when they need to exchange data on input and output parameters or to resolve conflicts on confluent vertices (associated with parallelism.)

Second, the local query translation scheme was verified by using various forms of queries. Specifically, a subquery (i.e., task) for CE was translated into a retrieval query similar to that for a database, while a subquery for SE or ME was mapped into a set of initial data.

Third, to execute a cycle (in the iterative fashion), an entire inferencing chain in each component system involved in the cycle had to be performed in each iteration even though the portions of the chain outside the cycle are not used except in the last iteration. This is an inevitable inefficiency because the nature of rule-based systems works in such a way, and we cannot change their internal inferencing

processes.

Fourth, we also confirmed that preexisting expert systems need not be compromised with respect to their original inferencing mechanisms for this rule-level integration, except modifying some of their rules for I/O redirections.

In conclusion, we demonstrated that our integrated system can produce optimal global answers for global queries based on the independent partial answers derived by heterogeneous rule-based systems. This methodology of problem solving is analogous to the interactive way in which a group of human experts collaborate to solve complex problems in their common domain. Specifically, we observed that independently-derived, thus probably conflicting, local answers can be resolved at right points in the inferencing chains corresponding to confluent vertices. Also, we showed that various data involved in local answers that are interdependent in an intertwined manner are systematically assembled into a global answer. Without our integrated system, obtaining a global answer would require ad hoc assemblies of partial (maybe incompatible) answers, which would require extensive human expertise. Still, the global answers from such assemblies are not guaranteed to be globally optimal.

CHAPTER IX

SUMMARY AND FUTURE RESEARCH

9.1 Summary

We have developed an approach to heterogeneous rule-based systems, whose data and knowledge are in the form of rules and constraints. This integration approach pursues a tight integration of such knowledge derivation systems. As opposed to an existing heterogeneous DDBMS, the component systems in our approach are not mere data retrieval systems. Rather, they derive data by applying rules and constraints to stored data. Our objective of this integrated system was to provide the user with a unified global view as if the heterogeneous rule-based systems were not independent systems, but were one system in the same way as heterogeneous database systems can be used as one central database system. Specifically, the rules and constraints can in effect be shared among heterogeneous component systems regardless of the incompatibility in their inferencing mechanisms and interpretation mechanisms.

To embody this objective, our system architecture is

based on a hybrid central control to allow the component systems to maintain their autonomies after the integration. We developed three major system components: a knowledge representation scheme, a global knowledge manipulation language, and a distributed query processing mechanism. To handle knowledge as opposed to data, a new knowledge representation scheme was developed to capture the dynamic aspects of knowledge as well as its conventional static aspects. The knowledge representation scheme was designed to be neutral to the control mechanisms of particular component systems. As a result, the best rules can always be selected regardless of their association with component systems with respect to a complex global query beyond any one component system's knowledge. The global knowledge manipulation language (GKML) has a declarative nature with sophisticated semantics. The focus of the GKML is to allow the user to observe the behavior of component rule-based systems from a global viewpoint. Specifically, it supports some hypothetic queries and repetitive queries. Consequently, the GKML has a high expressive power. The distributed query processing mechanism was developed to process the queries according to the semantics of the GKML.

A notable feature of our approach is to allow the rules and constraints to be shared in effect among heterogeneous component systems despite incompatibilities in their internal control mechanisms. That is, the integration among

heterogeneous deductive systems can take place at the rule level, instead of at the system level. The resulting advantages of our approaches are: 1) the best rules for a global query are selected from, and exercised in, many different systems at each intermediate step of knowledge derivation process, so that the best possible answer derivable from the component systems can always be derived, and 2) a knowledge transparency (a level higher than the conventional distribution transparency [CER85]) is achieved in that the user need not know which rules and associated component systems are to be involved, or even the existence of the independent component systems. As a result, this integrated system allows heterogenous rule-based systems to, in effect, function as one super-intelligent rule-based system, which coherently exercises all the expertise that its component systems possess.

The schemes and algorithms have been varified through a prototype development. The prototype system was tested with three independent expert systems as its component systems. They are based on incompatible rule formats and inferencing mechanisms.

Our approach provides a systematic way of integrating heterogeneous expert systems and consequently reduces the knowledge brittleness plaguing individual rule-based systems (in their isolated use.) The brittleness of knowledge [JAC86] due to the unstructuredness has been a well-recognized

drawback of rule-based expert systems as artificial experts. One conceivable way to enhance the robustness of knowledge is to implant some "redundancy" in knowledge by integrating multiple expertises in a symbiotic way [JAC86]. The rationale behind this idea is that intrinsic defects of an isolated knowledge derivation system can be compensated by extrinsic knowledge systems, whose scopes are often beyond that of the original knowledge derivation system.

9.2 Future Research

In a complex environment, many different users would issue their queries at the same time. To serve multiple queries in a concurrent fashion, we need to incorporate many conventional issues that are involved in concurrent processing in DBMSs [CER85 and DAT86] in our integrated system. Our approach already addressed some parallelism in distributed query processing, which could be generalized further in the context of a multi-user system.

Our approach could apply to many types of information systems as its component systems. Possible information systems include DBMSs and general application software based on procedural paradigms. Recall the basis of our approach described in Chapter Three. According to the basis, virtually any type of information system can be accommodated by our

approach as long as its knowledge derivation path can be represented by articulated paths. Of course, the possible level of integration would vary according to the characteristics of knowledge (and user interfaces) of component systems [FRE85 and SHW87].

In integrated knowledge processing and data processing, the efficiency in information exchange among different types of information systems becomes another important issue. Many efforts to couple AI and DB technologies have been made mainly for the purpose of eliminating the impedance mismatch between two types of information systems [SHE89]. However, such coupling efforts have been limited to using Prolog to access relational DBMSs [JAR84 and IOA88], or providing a simple interface between commercial shell and relational DBMSs [BUE85 and ABA86]. As suggested in SHE89, such efforts should be extended to cover a wide variety of types of information systems for our approach to function as a comprehensive integrated system.

REFERENCES

[ABA86] Abarbanel R. and Williams M., "A Relational Representation for Knowledge Bases," Technical Report, Intellicorp, Mountain View, CA, Apr. 1986.

[AHO74] Aho A., Hopcroft J. and Ullman J, The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading MA, 1974, pp. 176-195.

[AIK85] Aikins J., "Prototypical Knowledge for Expert Systems," AI Magazine, Vol. 20, pp.163-210.

[BAN87] Bannister R. and Moore M.,"General Rotational Machinery Expert System," IEEE Western Conf. on Expert Systems, 1987, pp.140-151

[BAT86] Batini C., Lenzerini M. and Navathe S.,"A Comparative Analysis of Methodologies for Database Schema Integration," ACM Computing Survey, Vol.18, No.4, Dec. 1986, pp.323-364.

[BEL82] Bell J.,"Data Modelling of Scientific Simulation Programs," ACM Int'l Conf. on Management of Data, SIGMOD, 1982, pp.79-86

[BON87] Bond A. and Ahmed S.,"Rule-Based Automatic Dimensioning," IEEE Western Conf. on Expert Systems, 1987, pp.128-135

[BOR84] Bordie M. and Jarke M.,"On Integrating Logic Programming and Databases," Proc. First Int'l Workshop on Expert Database Systems, Kiawah Island, SC, 1984.

[BRA88] Brazile R. and Swigger K., "GATES," IEEE Expert, Summer 1988, pp. 33-39.

[BRO81] Brodie M.,"On Modeling Behavioral Semantics of Databases," Proc. 7th VLDB, 1981, pp.32-42

[BRO85] Brownston L., Farrell, R., Kant, E. and Martin, N., Programming Expert Systems in OPS5, Addision Wesley, Reading MA, 1985, pp.51-60.

[BUC85] Buchanan B. and Shortliffe E., Rule-Based Expert Systems, Addison-Wesley, Reading MA, 1985, pp.51-70.

[BUE85] Buer D., "The Flexible Deductive Engine: An Environment for Prototyping Knowledge Based

Systems," Proc. of the Nineth International Joint Conference on Artificila Intelligence, Los Angeles, CA, Aug. 1985.

[CER85] Ceri S., Pelagatti G., Distributed Databases-Principles and Systems, McGraw-Hill, New York, 1985, pp.361-385.

[CHAL86] Chalfan K., "A Knowledge System That Integrates Heterogeneous Software for a Design Application," AI Magazine, Summer, 1986, pp.80-84.

[CHAN86] Chandrasekaran B.,"Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design," IEEE Expert, Fall 1986, pp.23-32

[COD79] Codd, E.F.,"Extending Database Relations to Capture More Meaning," ACM TODS, 4 4, 1979, pp. 397-434.

[COD82] Codd, E, "Relational Database: A Practical Foundation for Productivity," Comm. ACM, 25 2, 1982, pp. 109-118.

[CZE87] Czejdo B.,"An Approach to Schema Integration and Query Formulation in Federated Database Systems," Proceedings of the Third International Conference on Data Engineering, 1987, pp. 36-45.

[DAT86] Date C., Introduction to Database Systems, 4th edition, Addison-Wesley, Reading, MA, 1986.

[DEM89] DeMichiel L., "Performing Operations over Mismatched Domains," Proceedings of the Fifth International Conference on Data Engineering, Feb. 1989, Los Angeles, CA, pp. 36-45.

[DEN86] Dennings P.,"Towards a Science of Expert Systems," IEEE Expert, Summer 1986, pp.80-83.

[DWY83] Dwyer P. and Hevner A., "Transaction Optimization in a Distributed Database Testbed System," Proc. of IEEE compsac, 1983.

[ELM79] Elmasari R. and Wiederhold G., "Data Model Integration Using the Structural Model," Proc. ACM SIGMOD Conference, May 1979, pp. 319-326.

[FER83] Ferrier A. and Stangret C., "Heterogeneity in the Distributed Database Management System SRiUS-DELTA," Eighth VLDB, Mexico City, 1983.

[FIS86] Fisher E., "An AI-Based Methodology for Factory

Design," AI Magazine, Fall 1986, pp. 72-85.

[FRE85] Freksa C., "Knowledge Representation for Interactive Aircraft Design," Proc. of the Tech. Assessment and Management Conf. of the Gottlieb Duttweiler Institute, Apr. 1985, pp. 221-230.

[GER82] Gerring P. and Shortliffe E., "The Interviewer/Reasoner Model: An Approach to Improving System Responsiveness in Interactive AI Systems," AI Magazine, Fall 1982, pp.24-27

[GOL83] Golshani F., Maibaum T. and Sadler M.,"A Modal System of Algebras for Database Specification and Query/Update Language Support," Proc. of 9th VLDB, 1983, pp. 331-339.

[GOL84] Golshani F.,"Tools for the Construction of Expert Database Systems," Proc. First Int'l Workshop on Expert Database Systems 1984, pp. 442-455.

[HAM81] Hammer M. and McLeod D., "Database Description with SDM: A semantic database model," ACM Trans. on Database Systems, 6,3 (sept.), 1981, pp. 351-386.

[HAS82] Hass L., "R*: A Research Project on Distributed Relational DBMS," Database Engineering, Vol. 5, No. 4, 1982.

[HEI82] Heimbigner D.,"A Federated Architecture for Database Systems," Ph.D Dissertation, Univ. of Southern California, Los Angeles, CA, Aug. 1982.

[HEI85] Heimbigner D. and Mcleod D.,"A Federated Architecture for Information Management," ACM TOOIS, Vol.3, No.3, July 1985.

[HEL88] Held P.,"Extending the Capabilities of Database Management Systems to Support the Needs of Expert Systems," Ph.D. Dissertation, Univ. of Minnesota, MI, 1988.

[HUL87] Hull R. and King R., "Semantic Database Modelling: Survey, Applications, and Research Issues," ACM Computing Surveys, Vol. 19, No.3, September 1987, pp.201-260.

[IOA88] Ioannidis Y., "BERMUDA -- An Architectural Perspective on Interfacing Prolog to a Database Machine," in Proc. of the Second Int'l Conference on Expert Database Systems, Apr. 1988.

[JAC86] Jackson P., Introduction to Expert Systems, Addison-Wesley, Wokingham England, 1986, pp. 29-51.

[JAR84] Jarke M. and Vassiliou Y.,"Coupling Expert Systems and Database Management Systems," in Artificial Intelligence Applications for Business, Ablex, Norwood, NJ, 1984.

[KAE86] Kaemmer W. and Larson J.,"A Graph-Oriented Representation and Unification Technique for Automatic Selecting and Invoking Software Functions," Proc. of AAAI, 1986. pp. 825-830.

[KEL82] Kellog C.,"Knowledge Management: A Practical Amalgam of Knowledge and Database Technology," Proc. of the National Conf. on Artificial Intelligence, Carnegie-Mellon Univ., Pennsylvania PA, 1982.

[KEM87] Kemper A., Lockermann P. and Wallrath M., "An Object-Oriented Database System for Engineering Applications," ACM SIGMOD Record 1987, pp.299-310.

[KERN84] Kernighan B. and Pike R., The UNIX Programming Environment, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1984, pp. 29-31.

[KERS84] Kerschberg L. and Shepherd A.,"Constraint Management in Expert Database Systems," Proc. of the First Int'l Workshop on Expert Database Systems, Kiawah Island, SC, 1984.

[KRI87] Krishnamurthy V. et al.,"A Distributed Database Architecture for an Integrated Manufacturing Facility," Proc. of the Conf. on Data and Knowledge Systems for Engeering and Manufacturing, CT, 1987.

[LAN77] Landers T. and Rosenberg R., "An Overview of MULTIBASE," Distributed Databases, H.Schneider ed., North-Holland, 1977.

[LAR85] Larson J.,"ATOZ -- A Prototype Intelligent Interface to Multiple Information Systems," Proc. IFIP Working Conference, Rome, Sep. 1985.

[LEL88] Leler, W, Constraint Programming Language: Their Specification and Generation, Addison-Wesley Publishing Company, Reading MA, Inc. 1988, pp.15-38.

[LES83] Lesser V. and Corkill D., "The Distributed Vehicle Monitoring Testbed," AI Magazine, Fall 1983, pp.15-

33.

[LI88] Li Y.,"DKM - A Distributed Knowledge Representation Framework," Proc. of Second Int'l Conference on Expert Database System, 1988, pp.143-152.

[LYN84] Lyngbaek P. and McLeod D.,"Object Sharing in Distributed Information Systems," ACM TOOIS, Vol.2, No.2, Apr. 1984, pp.96-122.

[MAR84] Martin R.,"Integrating Database and Program Descriptions Using an ER Dictionary," Journal of Systems and Software, Vol.4, No.2 and 3, Jul. 1984, pp. 185-195.

[MCD82] McDermott, J.,"R1: A Rule-Based Configurer of Computer Systems," Artificial Intelligence, 19, 1982, pp.39-88.

[MIS84] Missikoff M.,"Toward Unified Approach to Expert and Database Systems," Proc. First Int'l Workshop on Expert Database Systems, Kiawah Island, SC, 1984.

[NAV89] Navathe S., Gala S. and Geum S.,"A Federated Approach to Loosely-Coupled Integration of Multiple Information Systems," Univ. of Florida, DBS R/D Center, Tech. Report, Apr. 1989.

[NII86] Nii H., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," AI Magazine, Summer 1986, pp. 38-53.

[NIL80] Nilson N., Principles of Artificial Intelligence, Morgan Kaufmann Publishers, Inc., Princeton NJ, 1980, pp. 131-191.

[PET85] Peterson J. and Silberschatz A., Operating System Concepts, 2nd edition, Addison-Wesley Co., Reading MA, 1985, pp. 307-308.

[PRA87] Pravier P.,"Honeyshell," Masters Thesis, Univ. of Florida, 1987.

[PIR89] Pirotte A. and Roelants D., "Constraints for Improving the Generation of Intensional Answers in a Deductive Database," Proc. of Fifth International Conference on Data Engineering, Los Angeles, Feb. 1989, pp.652-659.

[QUA87] Quan A. and Smith R.,"An Expert System for the Design of Space Shuttle Ascent Trajectory Throttle

Bucket," IEEE Western Conf. on Expert Systems, 1987, pp.65-72

[RAM88] Ramnarayan R. and Lu H., "A Data/Knowledge Base Management Testbed and Experimental Results on Data/Knowledge base Query and Update Processing," Proc. of ACM SIGMOD Int'l Conf. MOD, 1988, pp. 387-395.

[RAS87] Raschid L.,"The Design and Implementation Techniques for an Integrated Knowledge Base Management System," Ph.D. Dissertation, Dept. of Electrical Eng., Univ. of Florida, FL, 1987.

[RAS85] Raschid L. and Su S.,"Incorporating Knowledge Rules in a Semantic Data Model: An Approach to Integrated Knowledge Management," IEEE Conference on Artificial Intelligence Applications, Miami, Florida, 1985.

[RIC83] Riche E., Artificial Intelligence, McGraw-Hill Book Company, Singapore, 1983, pp. 247-294.

[ROM85] Roman G.,"A Taxonomy of Current Issues in Requirements Engineering," IEEE Computer, April 1985, pp. 15-21.

[ROT80] Rothnie J., Bernstein P., Fox S, Goodman N., Hammer M., Landers T., Reeve C., Shipman D. and Wong E., "Introduction to a System for Distributed Databases(SDD-1)," ACM TODS, Vol. 5, No. 1, 1980, pp. 1-17.

[SAK83] Sakai H.,"A Method for Entity-Relationship Behavior Modeling," Entity-Relationship Approach to Software Engineering, C.Daviset al.(eds), Elsevier Science Publishers B.V. (North-Holland) pp. 111-129

[SAT] Sathi A., Morton T and Roth S.,"An Intelligent Project Management System," AI Magazine (7)5, Winter 1986, pp.34-52.

[SCH85] Schutzer D.,"Artificial Intelligence-Based Very Large Data Base Organization and Management," Applications in Artificial Intelligence, Andriole S. (eds.), Petrocelli Books, Inc., Princeton, N.J., 1985.

[SED83] Sedgewick R., Algorithms, Addison-Wesley Inc., Reading MA, 1983, pp. 373-456.

[SHE89] Sheth A., "Does Loose AI-DBMS Coupling Stand a Chance?," Proc. of International Conference on Data

Engineering, Los Angeles, 1989, pp.252-254.

[SHI81] Shipman D., "The Functional Data Model and the Data Language DAPLEX," ACM TODS 6, No.1, 1981.

[SHO76] Shortliffe E., Computer-Based Medical Consultations: MYCIN, Elsevier, New York, 1976.

[SHW87] Shwartz S., Applied Natural Language Processing, Petrocelli Books, Princeton, NJ, 1987, pp. 237-256.

[SMI77] Smith J. and Smith D., "Database Abstractions: Aggregation and Generalization," ACM Trans. Database Systems, 2. 2 (June), pp. 105-133.

[SMI81] Smith R. and Davis N.,"Frameworks for Cooperation in Distributed Problem Solving," IEEE Trans. Systems, Man, Cybernetics, Vol. SMC-11, No.1, Jan. 1981.

[SRID87] Sridharan N.,"1986 Workshop on Distributed AI," AI Magazine, Fall 1987, pp.75-85

[SRIN87] Srinivas K., "COOL:An Expert System for Selection and Design of Heat Exchange Equipment," IEEE Western Conf. on Expert Systems, 1987, pp. 177-182.

[STE86] Sterling L. and Shapiro E., The Art of Prolog, The MIT Press, Cambridge MA, 1986, pp. 137-138.

[STO76] Stonebraker M., Wong E., Kreps P. and Held G., "The Design and Implementation of INGRES," ACM TODS 1, No. 3, Sep. 1976, pp. 189-222.

[STO86] Stonebraker M., "The Design of Postgres," Proc. 1986 ACM SIGMOD Conference on Management of Data, Washington, D.C., May 1986

[STO87] Stonebraker M., Hanson E. and Potanianos S., "The Design of the POSTGRES Rules System," Proc. 3rd Int'l Conference on Data Engineering, Feb. 1987, Los Angeles CA.

[SU85] Su S. and Raschid L.,"Incorporating Knowledge Rules in a Semantic Data Model: An Approach to Integrated Knowledge Management," IEEE Conference on Artificial Intelligence Applications, Miami, Florida, Dec. 1985.

[SU86] Su S.,"Modelling Integrated Manufacturing Data with SAM*," IEEE Computer, Jan. 1986, pp. 34-49.

[SU88] Su S., Lam H., Khatib M., Krishnamurthy V., Kumar A., Malik S., Mitchell M. and Barkmeyer E.,"The Architecture and Prototype Implementation of an Integrated Manufacturing Database System," Proc. of the COMPCON, Spring, 1988.

[SU90] Su S., Krishnamurthy V. and Lam H.,"An Object-Oriented Semantic Association Model (OSAM*)," in Artificial Intelligence: Manufacturing Theory and Practice, Kumara S., Soyster A., and Kashyap R. (eds.), Industrial Engineering and Mangement Press, Narcross, GA, 1990.

[TAI86] Taie M. and Srihari S.,"Device Modeling for Fault Diagnosis," Expert Systems in Government Symposium, 1986, pp. 144-150.

[TANE88] Tanenbaum A., Computer Networks, Second Edition, Prentice-Hall, Inc., Englewood Cliffs NJ, 1988, pp. 370-439.

[TANG86] Tangen K. and Wretling U.,"Intelligent Front Ends to Numerical Simulation Programs," Proceedings of 6th Technical Conf. of British Computer Society (Expert Systems) Dec. 1986, pp. 254-265.

[THO86] Thompson B., "Knowledge + Control = Expert Systems," AI Expert, Vol. 1, No. 3, Nov. 1986, pp. 25-29.

[TOM86] Tomada D.,"A Framework of Expert System with Strategic Knowledge," Proc. of Int'l Conf. on Data Engineering, IEEE, 1986, pp.236-243.

[TRO85] Troeder C. and Naumann H.,"Expert Systems of Optimal Selection of Machine Elements," Proc. of the Tech. Assessment and Management Conf. of the Gottlieb Duttweiler Institute, Apr. 1985, pp. 207-214.

[TSI76] Tsichritzis D. and Lochovsky F., "Hierarchical Database Management: A Survey," ACM Computing Survey, 8, No. 1, March 1976.

[TSU88] Tsur S., "LDL-A Technology for the Realization of Tightly Coupled Expert Database Systems," IEEE Expert, Fall 1988, pp.41-51.

[URS83] Ursprung P. and Zehnder C., "HIQUEL: An Interactive Query Language to Define and Use Hierarchies," in Entity-Relationship Approach to Software Engineering, Davis et al(eds.), 1983, pp. 299-316.

[VAS85] Vassiliou Y., "Integrating Database Management and

Expert Systems," Proceedings of the Conference on Database Systems for Office, Engineering and Science, Karlsruhe, West Germany, March 1985.

[WIE84] Wiederhold G. and Missikoff M., "Toward a Unified Approach to Expert and Database Systemss," Proceedings of the First International Workshop on Expert Database Systems, October 1984.

[WIL84] Wilson G.,"Distributed Database Considerations in an Expert System for Radar Analysis Intelligent Database Interfaces," Proc. First Int'l Workshop on Expert Database Systems, 1984, pp. 586-602.

[WON84] Wong E.,"Enhancing INGRES with Deductive Power," Proc. of the First Int'l Workshop on Expert Database Systems, Kiawah Island, South Carolina, 1984.

[ZAN83] Zaniolo C., "The Data Language GEM," ACM SIGMOD Record Vol. 13, No.4, 1983, pp. 207-217.

[ZAN85] Zaniolo C.,"The Representation and Deductive Retrieval of Complex Objects," Proc. 11th Int'l Conf. on VLDB, 1985.

APPENDIX

1. Syntax of the Global Query Language

(* denotes zero or more occurrences, except ones without arguments with its literal meaning. + denotes one or more occurrences.)

Types I and II

```
GET <entity>* WHEN <source condition>+

<source condition>  : <entity> = <constant>
                    | <entity> = <entity> <relop> <constant>
<relop>             : PLUS | MINUS | TIMES | DIVIDE
<entity>            : object class | attribute | operation
```

Type III

```
SET <initial condition>+ FOR <target condition>+

<initial condition> : <entity> [= <constant>] [BY]
<target condition>  :   <entity>   <comop>   <constant>
[WITHIN]
<entity>            : object class | attribute | operation
<comop>             : = | != | > | < | =< | >=
```

Type IV

```
GET <entity>+ FOR *

<entity>            : object class | attribute | operation
```

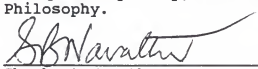
Biographical Sketch

Born in Korea, on Feb. 4, 1956, the author attended Pusan High School in Pusan. He received a Bachelor of Science degree in electrical engineering from Seoul National University, in Feb. 1979. He was awarded the degree of Master of Engineering in electrical engineering by the Korea Advanced Institute of Science and Technology in Seoul, in Feb. 1981. He will receive the degree of Doctor of Philosophy in electrical engineering in August, 1990.

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

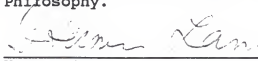
Stanley Y.W. Su, Chair
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



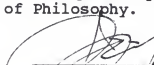
Shamkant Navathe
Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Herman Lam
Associate Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.



Jose C. Principe
Associate Professor of Electrical Engineering

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Paul Fishwick
Assistant Professor
of Computer and Information Sciences

This dissertation was submitted to the Graduate Faculty of the College of Engineering and to the Graduate School and was accepted as partial fulfillment of the requirements for the degree of Doctor of Philosophy.

August, 1990

Winfred M. Phillips
Dean, College of Engineering

Madelyn M. Lockhart
Dean, Graduate School